View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Illinois Digital Environment for Access to Learning and Scholarship Repository

UNIVERSITY OF ILLINOIS

May 2 19 90

THIS IS TO CERTIFY THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

Daniel J. Kaiser

ENTITLED A Parallel Equation-Based Algorithm for Simulating

Single Separation Columns

IS APPROVED BY ME AS FULFILLING THIS PART OF THE REQUIREMENTS FOR THE

DEGREE OF Bachelor of Science in Chemical Engineering

Mark Stadtherr

Instructor in Charge

Approved: A J McHugh

HEAD OF DEPARTMENT OF Chemical Engineering

©1954

A Parallel Equation-Based Algorithm for Simulating Single Separation Columns

By

Daniel J. Kaiser

Thesis

for the
Degree of Bachelor of Science
in
Chemical Engineering

College of Liberal Arts and Sciences
University of Illinois
Urbana, Illinois

1990

ABSTRACT

A parallel equation-based algorithm has been developed to simulate single separation units. This algorithm was coded in FORTRAN and optimized for use on the Cray-2 Supercomputer, which has four processors available for multitasking. The algorithm was tested on several problems, and was generally reliable in converging to the correct solution. From the test results, the parallel program achieved speedups in excess of a factor of three over single processor operation. In addition, it was found that this algorithm, even on one processor, is inherently more efficient than normal Gauss-Jordan elimination. This new parallel algorithm clearly demonstrates that it can offer a much faster means for simulating distillation and absorber columns than traditional methods.

ACKNOWLEDGEMENTS

I would like to thank Dr. Stadtherr for giving me the opportunity to work with him and his research group, and for providing me not only with a challenging and interesting thesis topic, but one which was within the scope of my abilities and time limitations.

I would also like to thank John O'Neill for guiding my progress in this work, and Carol Schnepper for teaching me many many things about computers. And I would like to thank the entire research group for their advice and support in accomplishing this project.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

Process flowsheeting, the use of computers to design and simulate chemical processes, has found extensive use in industry recently, as many commercial software packages have become available. These packages save the engineer time and also reduce human error by performing many of the tedious calculations previously done by hand. Also, flowsheeting packages allow the design engineer to "experiment" with a process by simply changing a design variable and letting the computer recalculate all the other variables. Of course, the faster the software can run, the easier it is for the engineer to achieve these goals, and speed depends on the computer technology available, which fortunately is evolving new architectures that are constantly increasing in both clockspeed and memory. As computer architectures evolve, new algorithms must be developed to exploit their increasing capabilities.

With mainframe computers, new technologies have resulted in two important advances: (1) vector processors, and (2) multi-processor machines. These powerful architectures, if properly used, can significantly decrease the run-time of many programs. As of now, these architectures are currently found only on the most advanced mainframes, such as the Cray supercomputers. In time, though, these types of computers will becomes less expensive and gain wider acceptance, finding their way into the mainstream of commercial computers. This will result in the need for a new generation of software to take advantage of vector and multi-processor machines. As computers evolve, so must the software.

This study focuses specifically on developing a new computer algorithm for the optimization of single separation units, distillation and

absorber columns, by using an equation-based method founded on the Newton-Raphson technique. However, what differentiates this algorithm from others is its attempt to utilize the advantages of a multi-processor machine. The goal therefore becomes constructing the program in such a way that a large portion of it can be "divided up" among the various processors, allowing them to work concurrently. Theoretically, the maximum speedup that can be achieved is a factor of n, where n is the number of processors available. This study determines how close one can approach this theoretical limit with a distillation algorithm written for the Cray-2 supercomputer.

II. ADVANCED COMPUTER ARCHITECTURES

In most computers, the processor handles instructions one part at a time. For example, consider the following DO loop:

```
      do 10 i = 1,100
         A(i) = B(i) + C(i)
10    continue
```

A normal processor will execute this loop in the following manner:

1. Fetch B(1) from storage
2. Fetch C(1) from storage
3. Add B(1) and C(1)
4. Store the result in A(1)
5. Increment i
6. Repeat

On the other hand, a vector processor, such as in the Cray X-MP, can do the same task in a much more efficient manner:

1. Fetch all of vector B from storage
2. At the same time, fetch vector C from storage
3. As soon as the first elements of both vectors arrive at the processor, add them together
4. As soon as addition is complete, store the result in A
5. Continue until the loop is completed

This process is illustrated in Figure 1:

Fortunately, the FORTRAN compilers on the Cray are capable of identifying and vectorizing such code; no special directives are required from the programmer, generally. However, the programmer must take care to avoid writing a portion of code that cannot vectorize. According to Levesque and Williamson (pp. 90-91), things to avoid include:

1. Recursion
2. Subroutine calls
3. References to external functions unknown to FORTRAN 77
4. Input/output statements
5. Assigned GOTO statements
6. Some nested IF blocks

7. GOTO statements that exit the loop
8. Backward transfers within a loop

If a programmer keeps these few items in mind while programming, he or she can easily develop code that the compiler can vectorize.

Another advanced architecture, with which this study is primarily concerned, is multi-processor machines. Like vector processing, it provides a means for completing several tasks simultaneously. However it differs in the granularity (or level) with which it does tasks simultaneously. For example, consider the following DO loop:

```
      do 20 i = 1,256
         A(i) = B(i) + C(i)
20    continue
```

If there are four processors available, each can be assigned 64 iterations of the loop, and since each iteration is completely independent of the others, the processors can operate concurrently. And, of course, each processor can also be carrying on vector processing as well. Parallelism on the DO loop level, such as this, is known as microtasking (Cray Multitasking p. 4-1). Parallelism can occur at other levels as well. Consider the following piece of code:

```
call task1(arguments)
call task2(arguments)
call task3(arguments)
call task4(arguments)
```

Again if there are four processors available, and each subroutine is independent of the others, then these four subroutines can be run concurrently. Parallelism on the subroutine level, such as this, is known as macrotasking (Cray Multitasking p. 5-1).

Exploiting the use of several processors simultaneously requires much more programmer attention than does vector processing. Essentially, it is up to the programmer to develop an algorithm that will parallelize efficiently. Then once the algorithm is coded, special compiler directives or subroutine calls must be inserted to coordinate the operation of the processors. Currently, Cray is working on developing compilers that can recognize parallel code and automatically insert the correct directives (known as autotasking). However, these compilers are still in their developmental stages, and even when fully mature, do not eliminate the programmer's responsibility to develop code that can efficiently parallelize.

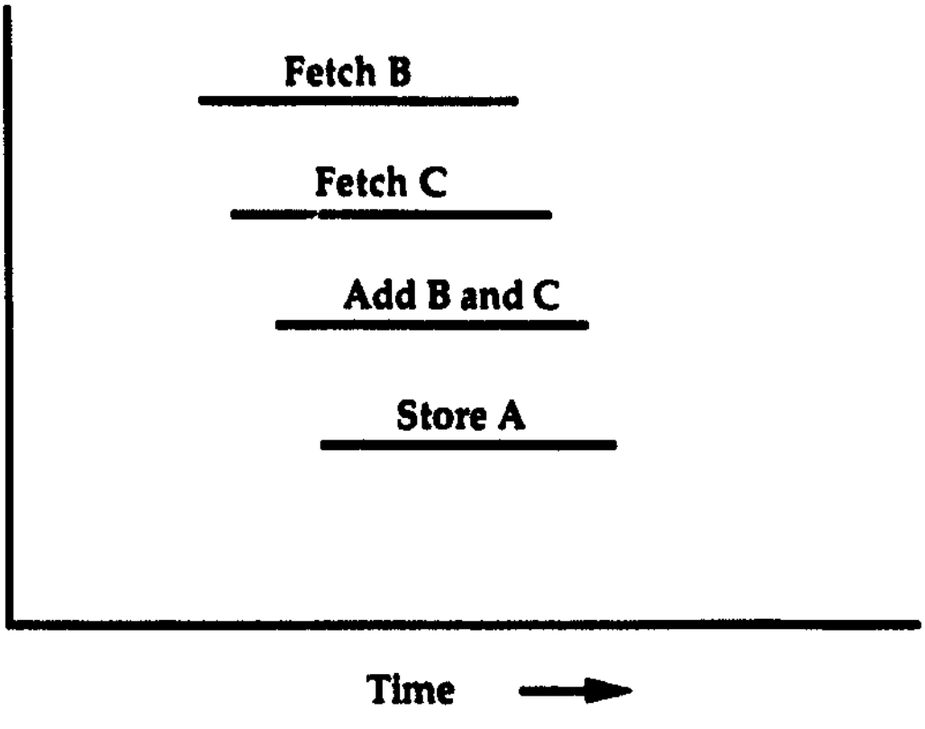

Figure 1

Timing Diagram of Vectorized Loop

(Levesque and Williamson p. 39-42)

III. PARALLEL EQUATION-BASED METHOD

The method used here to optimize separation units is the equation-based method, whereby a set of non-linear equations are solved simultaneously. The advantage of this method over the sequential modular approach, a step-by-step iterative technique, is its speed. However, it requires significantly more computer memory, but with today's advanced mainframes, such as the Cray, memory size is increasingly becoming an insignificant concern.

For separation problems, the equations are obtained by doing mass, energy, and equilibrium balances around each tray. This is the formulation of Naphtali and Sandholm (1971) and is shown below:

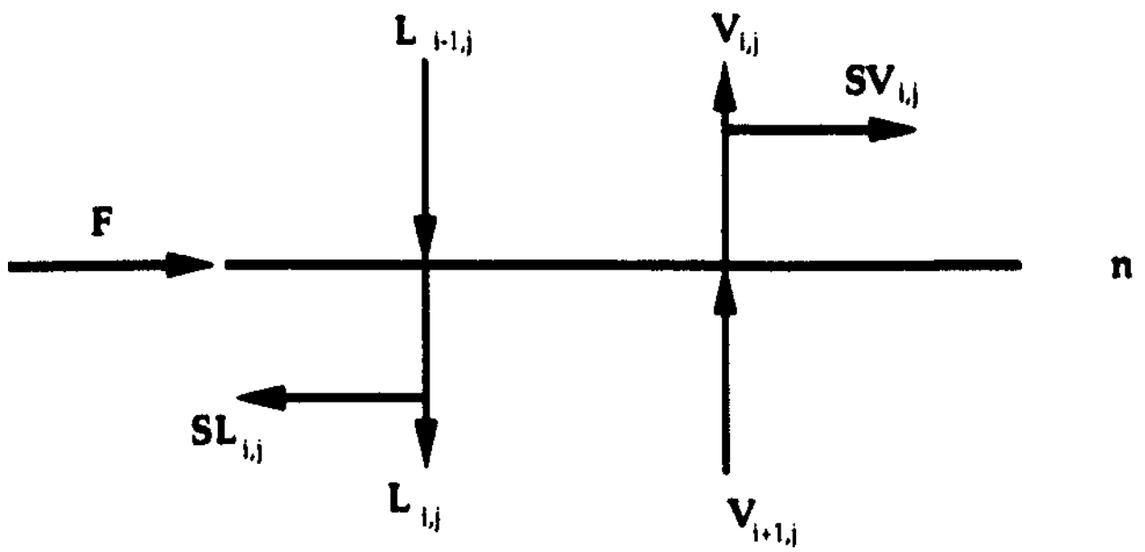

Figure 2
Tray Configuration

The equations that result from each tray are as follows:

Mass: $M_{i,j} = V_{i,j} + SV_{i,j} + L_{i,j} + SL_{i,j} - V_{i+1,j} - L_{i-1,j} - F_{i,j}$

Energy: $E_i = HV_{i,j} + HSV_{i,j} + HL_{i,j} + HSL_{i,j} - HV_{i+1,j} - HL_{i-1,j} - HF_{i,j}$

Equilibrium: $Q_{i,j} = n_i K_{ij} V_i L_{ij}/L_i - V_{ij} + (1-n_i) V_{i+1,j} V_i/V_{i+1}$

V = vapor flowrate
L = liquid flowrate
SL = liquid sidestream
SV = vapor sidestream

i = tray #
j = component #
K = equilibrium (y/x)
n = tray efficiency

The equations are set up as discrepancy functions, so that as the values of the variables approach the optimum, the functions will approach zero. Also, one should note that for each tray, there are j mass equations, j equilibrium equations, and one energy equation, for a total of 2j + 1 equations, and thus variables, per tray. Therefore, for the entire problem, there are (2j+1) x (# of trays) variables, a number known as the *order* of the problem. This formulation provides an extremely rigorous simulation of the column and allows for multiple feeds and sidestreams. For absorbers, these equations can be used for all trays, but for distillation columns, the energy equation for both the condenser and reboiler must be substituted with one of the following:

1.	specify Q	merely add Q to energy balance
2.	reflux/reboil ratio	L - V(R) or V- L(R)
3.	specify temp	T - Tspec
4.	specify flowrate	V - Vspec or L - Lspec

These substitutions are easy to incorporate and provide the user with several options from which to choose.

To solve all of these equations, the algorithm employs the Newton-Raphson method:

(1) $$\Delta x^{m+1} = - (J^{-1}|_{x^m})\, F^m$$

where x^m is the vector of variables after the m^{th} iteration, F^m is the vector of discrepancy functions evaluated at x^m, J^{-1} is the Jacobian matrix(dF/dx), and Δx^{m+1} is the update to the variables:

(2) $$x^{m+1} = x^m + f\,\Delta x^{m+1}$$

and f is a factor used to select the optimal fraction of the update vector.

Now, to evaluate the Jacobian, the algorithm must calculate a host of derivatives. Fortunately, the set of equations for each tray contains only variables from itself and the trays above and below it, which means that most of the elements of the Jacobian are zeroes. Overall, the Jacobian is constructed as shown in Figure 3. Actually, each element in the matrix is a *block* of elements. For example, the upper left element can be expanded as shown in Figure 4:

$$J = \begin{bmatrix} \frac{dF_1}{dX_1} & \frac{dF_1}{dX_2} & \cdots & \frac{dF_1}{dX_n} \\ \frac{dF_2}{dX_1} & \frac{dF_2}{dX_2} & \cdots & \frac{dF_2}{dX_n} \\ \vdots & & & \vdots \\ \frac{dF_n}{dX_1} & \frac{dF_n}{dX_2} & \cdots & \frac{dF_n}{dX_n} \end{bmatrix}$$

Figure 3

Jacobian Matrix

$$\frac{dF_1}{dX_1} =$$

	V_{11}	V_{12}	$\cdots$	V_{1k}	T_1	L_{11}	L_{12}	$\cdots$	L_{1k}
M_{11}	$\frac{dM_{11}}{dV_{11}}$	$\frac{dM_{11}}{dV_{12}}$	$\cdots$	$\frac{dM_{11}}{dV_{1k}}$	$\frac{dM_{11}}{dT_1}$	$\frac{dM_{11}}{dL_{11}}$	$\frac{dM_{11}}{dL_{12}}$	$\cdots$	$\frac{dM_{11}}{dL_{1k}}$
M_{12}	$\frac{dM_{12}}{dV_{11}}$	$\cdots$		$\frac{dM_{12}}{dV_{1k}}$	$\vdots$	$\frac{dM_{12}}{dL_{11}}$	$\cdots$		$\frac{dM_{12}}{dL_{1k}}$
$\vdots$	$\vdots$			$\vdots$		$\vdots$			$\vdots$
M_{1k}	$\frac{dM_{1k}}{dV_{11}}$	$\cdots$		$\frac{dM_{1k}}{dV_{1k}}$	$\frac{dM_{1k}}{dT_1}$	$\frac{dM_{1k}}{dL_{11}}$	$\cdots$		$\frac{dM_{1k}}{dL_{1k}}$
E_1	$\frac{dE_1}{dV_{11}}$	$\cdots$		$\frac{dE_1}{dV_{1k}}$	$\frac{dE_1}{dT_1}$	$\frac{dE_1}{dL_{11}}$	$\cdots$		$\frac{dE_1}{dL_{1k}}$
Q_{11}	$\frac{dQ_{11}}{dV_{11}}$	$\frac{dQ_{11}}{dV_{12}}$	$\cdots$	$\frac{dQ_{11}}{dV_{1k}}$	$\frac{dQ_{11}}{dT_1}$	$\frac{dQ_{11}}{dL_{11}}$	$\frac{dQ_{11}}{dL_{12}}$	$\cdots$	$\frac{dQ_{11}}{dL_{1k}}$
Q_{12}	$\frac{dQ_{12}}{dV_{11}}$	$\cdots$		$\frac{dQ_{12}}{dV_{1k}}$	$\vdots$	$\frac{dQ_{12}}{dL_{11}}$	$\cdots$		$\frac{dQ_{12}}{dL_{1k}}$
$\vdots$	$\vdots$			$\vdots$		$\vdots$			$\vdots$
Q_{1k}	$\frac{dQ_{1k}}{dV_{11}}$	$\cdots$		$\frac{dQ_{1k}}{dV_{1k}}$	$\frac{dQ_{1k}}{dT_1}$	$\frac{dQ_{1k}}{dL_{11}}$	$\cdots$		$\frac{dQ_{1k}}{dL_{1k}}$

Figure 4

Block Element of Jacobian

Evaluating a matrix of this size could be extremely expensive computationally. But as mentioned earlier, a vast majority of the blocks are actually zeroes. In fact, the Jacobian matrix takes on a very symmetric form, known as the block-tridiagonal form. It is this very fact that allows distillation problems to be solved in parallel, and how this is done will be discussed later. Figure 5 below shows the overall structure of the Jacobian once it has been evaluated:

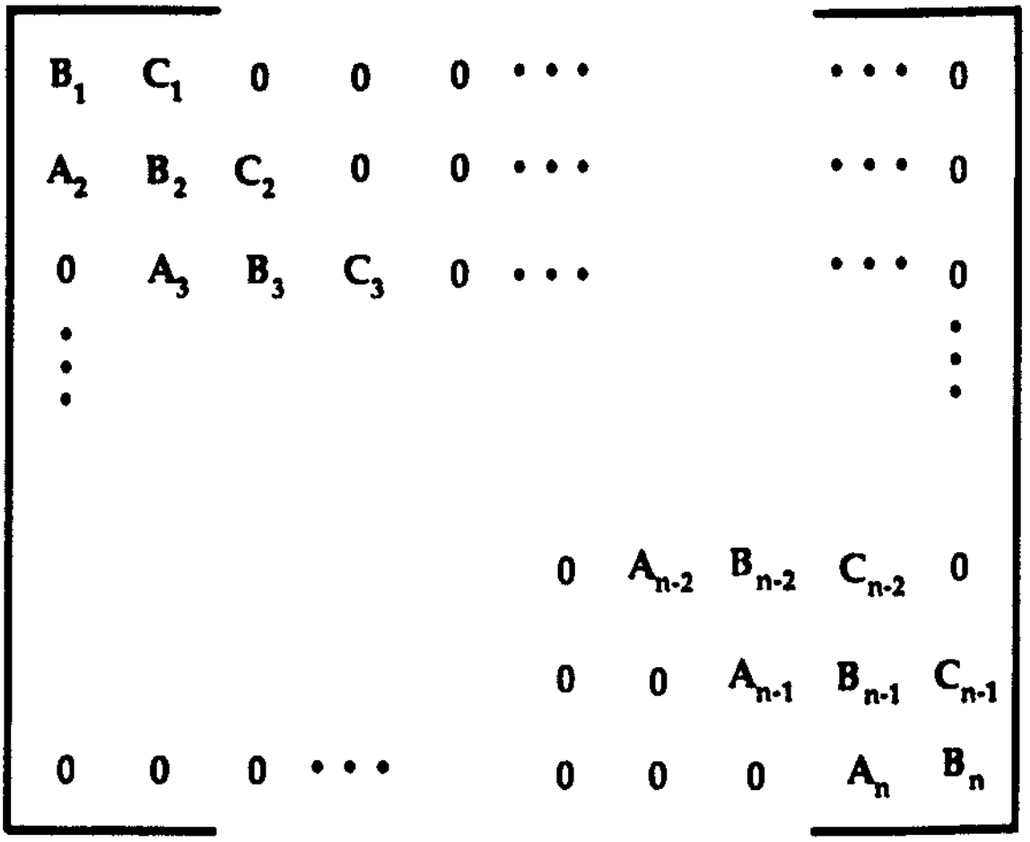

Figure 5
Evaluated Jacobian

Once the Jacobian has been evaluated, rearranging equation (1) and solving using Gauss-Jordan elimination will yield the values for the variable update vector Δx.

If one were to write a program to implement this entire method, one would find that it spends a vast majority of its time working on the Gauss-Jordan elimination step. Since this step is the main bottleneck in the program, if one can speed it up, then one can effectively speed up the entire program. To accomplish this, an algorithm developed by Sameh (p. 37-57) is

utilized to perform the Gauss-Jordan step in parallel. To illustrate how this algorithm works, say for example the computer has four processors available. The algorithm then partitions the Jacobian (Figure 5) into four different sections, each one the responsibility of a unique processor. The processors simultaneously perform Gauss-Jordan elimination, each operating on its own section, to produce the partially solved matrix shown in Figure 6 (note the asterisks represent fill-in which occurs during the Gauss-Jordan process and is why Figure 6 cannot be immediate solved for the answers):

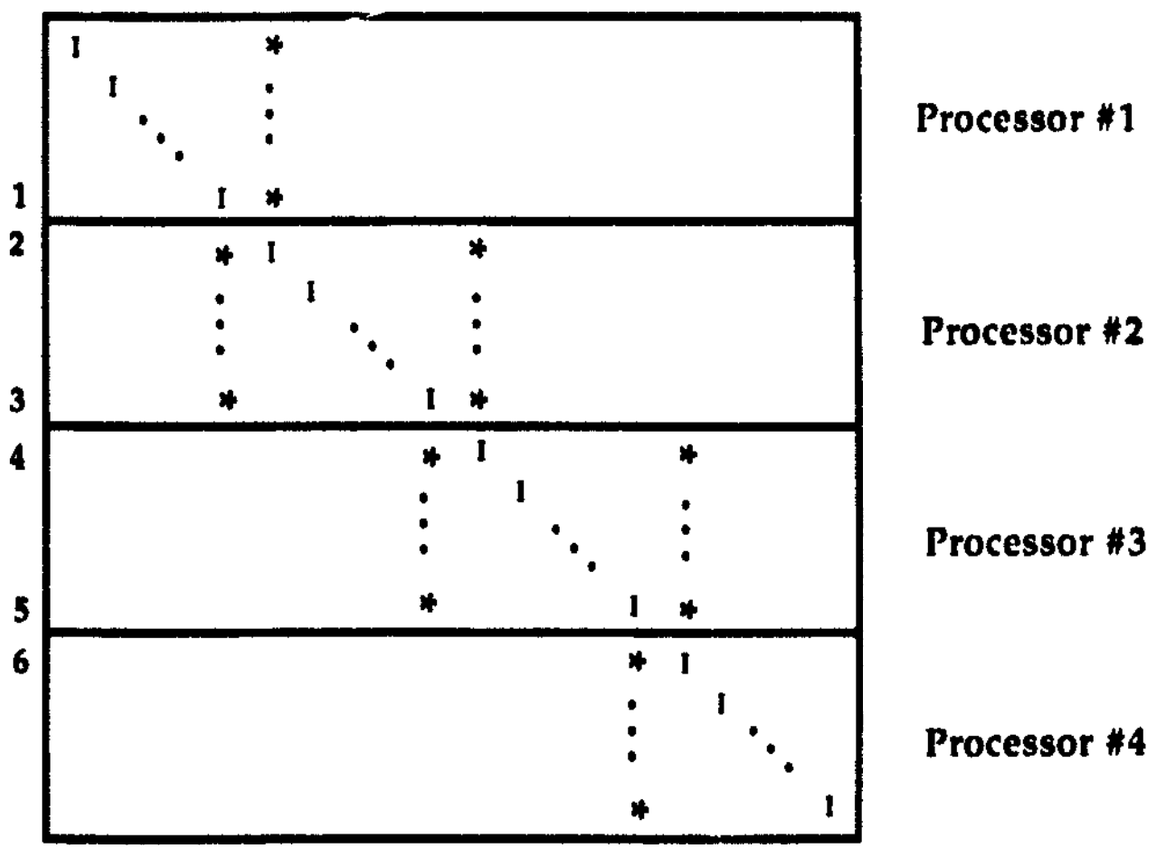

Figure 6
Results of Parallel Gaussian Elimination

Because of the fill-in that results, the system still cannot be solved. However, if one merely selects the rows of the matrix that border the processor partitions (numbered 1 through 6 in Figure 6) and place these rows in another matrix, they form a complete set of equations that can be solved explicitly (See Figure 7). Once the values for these variables are known, the remaining variables can be calculated with back-substitution on the matrix in Figure 6.

At this point one might be tempted to conclude that as the number of available processors increases, the execution time will increase. However this may be true only up to a certain point because as the number of equations increases, the size of the matrix in Figure 7 will also increase, and this matrix cannot be solved efficiently in parallel. As a result, this algorithm would seem best suited to run on a machine with a few very powerful processors rather than one with many less powerful processors. The Cray-2 fits the former category very well as it has four extremely fast processors (4-nanosecond clock cycle) and so will be the machine used in this study.

	1	2	3	4	5	6
1	I	*	0	0	0	0
2	*	I	0	*	0	0
3	*	0	I	*	0	0
4	0	0	*	I	0	*
5	0	0	*	0	I	*
6	0	0	0	0	*	I

Figure 7

Extracted Matrix

IV. COMPUTER PROGRAM

The computer program to implement this parallel algorithm is shown in Appendix A, but some discussion of how it works is necessary before moving on. The program has been written to be, relatively speaking, somewhat user friendly, which means that a person with knowledge on how the input files should be constructed can configure the program to solve many different types of problems simply by adjusting the dimensions of the arrays and adjusting the values contained in the PARAMETER statements. Actually, the program could have gone one step further in friendliness by simply overdimensioning the arrays to handle all reasonable types of problems and by reading in the values currently defined in PARAMETER statements. However, this was not done because these features would cause the program to run more slowly. If the arrays are overdimensioned, a small-size problem will run slowly because the computer must shuffle around large portions of arrays that are not even being used. The values defined in PARAMETER statements are there because they are frequently used as counter limits in DO loops, and the Cray FORTRAN compiler is better able to vectorize a DO loop when its length is exactly known. In other words, some user friendliness is sacrificed in order to maximize the speed of the program, but not to an extent where it is too cumbersome to use.

The main program first calls subroutine INPUT which reads in all the necessary data, such as initial guesses, thermodynamic data, and user options. Next, the program calls DISCREP which evaluates the discrepancy functions (F) using the current values for the variable vector (x). Then the program calls DERIV which evaluates the Jacobian using the current variable values. When completed, the program then enters the Gauss-Jordan elimination

phase. Depending on what option the user selects, the program will either do Gauss-Jordan elimination in a normal procedure (REGSOLVE) or it will begin the Sameh algorithm. For the Sameh algorithm, the first step is a call to PARTIAL which is the subroutine that executes Gauss-Jordan elimination in parallel until the matrix in Figure 6 is obtained. Once this step is complete, the program calls SELECT which collects the lines of the Jacobian matrix bordering the processor partitions and condenses them in the matrix shown in Figure 7. A call to FSTSOLVE then quickly solves this short set of equations. Finally, SCATTER1 and SCATTER2 take the update variables calculated by FSTSOLVE and perform back-substitution to obtain the remaining update variables, completing the Sameh Gauss-Jordan elimination algorithm.

At this point, the program calls SEARCH which selects the best fractional portion of the update vector. Recalling equation (2):

(2) $$x^{m+1} = x^m + f\,\Delta x^{m+1}$$

SEARCH determines the value for f (between 0 and 1) that will minimize the sum of the squares of the values of the discrepancy functions and then carries out equation (2) to calculate the new values for the variable vector. Next, the program calls REALVAR which checks to make sure that none of the new variables have passed into a physically meaningless region (such as a negative flowrate). If it finds such a value, it adjusts the value so that it no longer is unrealistic.

Now the program tests for convergence by first calling DISCREP to evaluate F with the new values for x and then SUM to determine the sum of the squares. If the value returned by SUM is less than some tolerance, or the program has completed a maximum number of iterations, the program exits

the iteration loop, prints out the final values (subroutine OUT), and quits. If another iteration is necessary, the program returns to the top for another run through the routines.

As mentioned before, there are two ways to implement parallel processing on the Cray-2--microtasking and macrotasking. This program utilizes microtasking and is implemented by inserting special compiler directives in the program. The program is then compiled using the following command:

CF77 -Zm filename

Inserting -Zm tells the compiler that microtasking compiler directives (CMIC$) are to be interpreted. If -Zm is left off, the compiler merely treats the directives as comment statements (since they begin with a "c"). In the program itself, the first directive, CMIC$ GETCPUS 4, appears in the main program very near the beginning, and instructs the computer to get all four CPUs focused on the program. However, even though all the processors have been "fetched" at this point, only one will be working on the program at any given time until PARTIAL is reached. The code for PARTIAL is preceded with the directive CMIC$ MICRO which tells the computer that this is a microtasked subroutine. PARTIAL itself then consists of one giant DO loop which iterates four times. The first iteration does Gauss-Jordan elimination on the top quarter, the second iteration on the second quarter and so on. Therefore, each iteration of this loop is completely independent of the others, which means the loop is ideal for parallelization. Inserting the directive CMIC$ DO GLOBAL just in front of the loop tells the computer this loop can be split up among the processors fetched by the GETCPUS directive.

Additionally, subroutine DERIV is also microtasked in much the same way since the evaluation of each tray is completely independent of each other. Implementing microtasking is actually not very difficult. All one needs to do is insert just a few compiler directive in the correct places. The compiler then takes care of the rest by interpreting these commands and actually inserting more lines into the program to make calls to the Cray microtasking functions and routines, which then guide the operation of the processors. Cray has made implementing microtasking easy, but they still have not taken away the responsibility of the programmer to construct code that will parallelize, which in this case means a DO loop with completely independent iterations. Microtasking can be easy, but the code must be properly written.

V. RESULTS AND DISCUSSION

Once the program was written and debugged, testing began to determine the robustness of the program. First, the program was tested on an absorber problem that was previously used by Napthali and Sandholm. The details of the problem are shown in Appendix B, but essentially it is an absorber tower with 20 trays and four components with a wide range in volatility. Applying the formulas derived earlier, this makes for a total of

$$(2 \times 4 + 1) \times 20 = 180 \text{ variables}$$

The program was able to solve this problem in five iterations to a sum of the squares of less than 1×10^{-15}. The compositions of the tops and bottoms streams are shown in Appendix B as well. Next, to test the programs ability to handle distillation columns, a distillation problem, again from Napthali and Sandholm, was used. This problem featured three hydrocarbons with similar volatilities, and twenty trays, with the top and bottom trays being a partial condenser and partial reboiler respectively. Furthermore, a sidestream was drawn off the condenser. Again applying the formulas, this amounts to

$$(2 \times 3 + 1) \times 20 = 140 \text{ variables}$$

This problem converged in 4 iterations to a sum of the squares of less than 1×10^{-10}. At this point it was clear the program was capable of handling both absorbers and distillation columns, as well as sidestreams, and wide ranges in volatility.

The next step was to collect data on the efficiency of the algorithm itself by determining the following relationships:

1. wallclock time and MFLOPS versus number of variables using one CPU

2. wallclock time and MFLOPS versus number of variables using four CPUs
3. wallclock time and MFLOPS versus number of variables using normal Gauss-Jordan elimination
4. wallclock time versus block size

Wallclock time is defined as the amount of processor time needed to execute the program in a dedicated environment using just one processor. This provides a very direct way of evaluating the speed of the program. MFLOPS (millions of floating point operations per second) also indicates the efficiency of the program by tallying the amount of "work" the computer does for a unit of time. The higher the MFLOPS, the more efficient the program.

To collect this data, the absorber problem was used for all runs to maintain uniformity. The first three relationships were obtained by varying the number of trays (and thus the number of variables) of the absorber, while the fourth relationship was obtained by adding more components to the absorber (and thus increasing the block size).

Figure 8 shows the wallclock results and Figure 9 the MFLOPS for the original problem (block size 9). Comparing the results on one CPU and four CPUs shows that a significant amount of speedup occurred from microtasking. This is shown in Figure 10. Clearly, the speedup increases with the number of variables, which is due to the fact that as the number of variables increases, the program spends a greater and greater fraction of its time in PARTIAL and DERIV, which are the microtasked routines. At the highest value, speedups of about about a factor of three occurred. Recall that the maximum theoretical speedup would be four, using four processors. In practice, a speedup of three is considered very good.

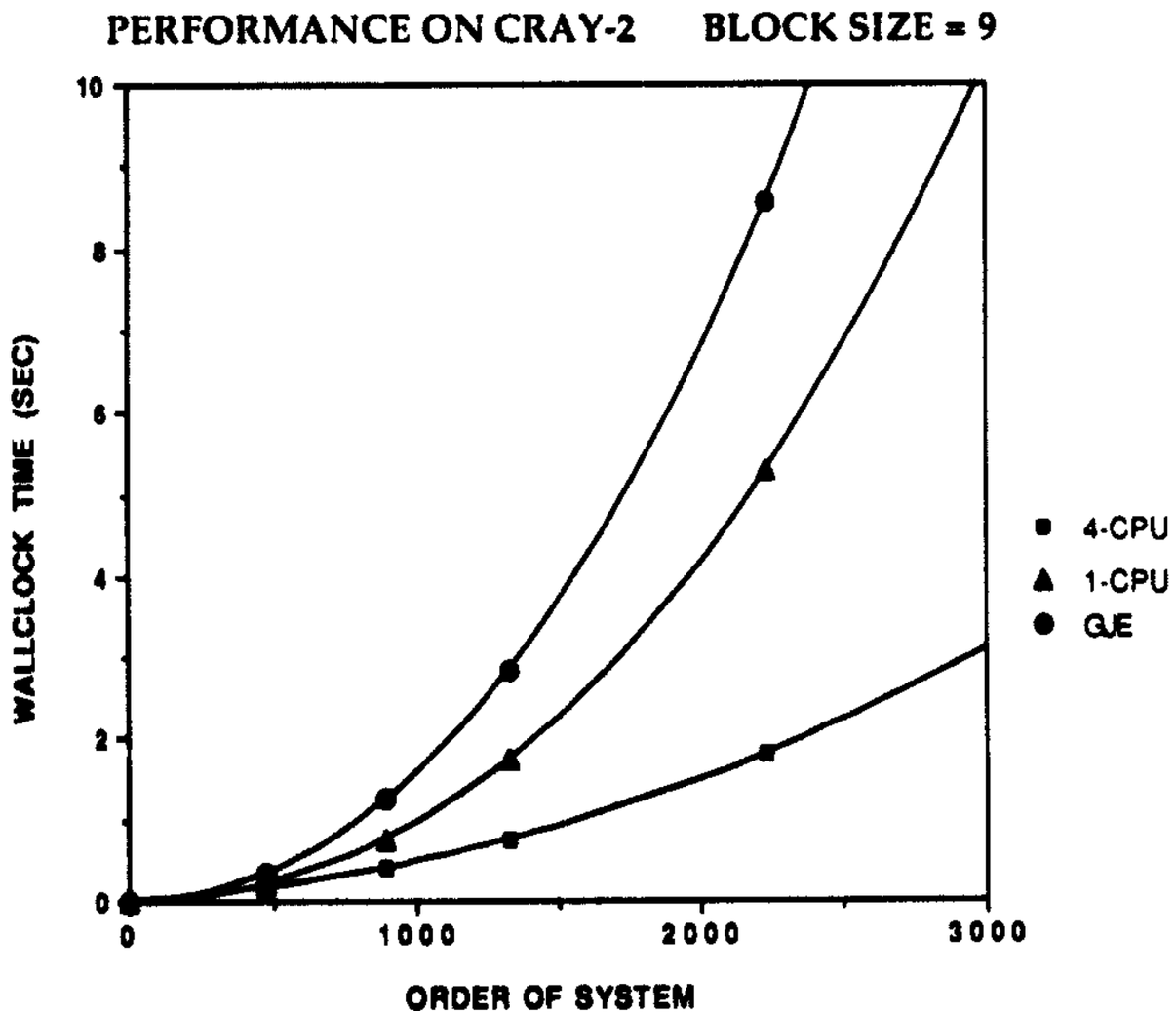

Figure 8
Wallclock Results for Blocksize = 9

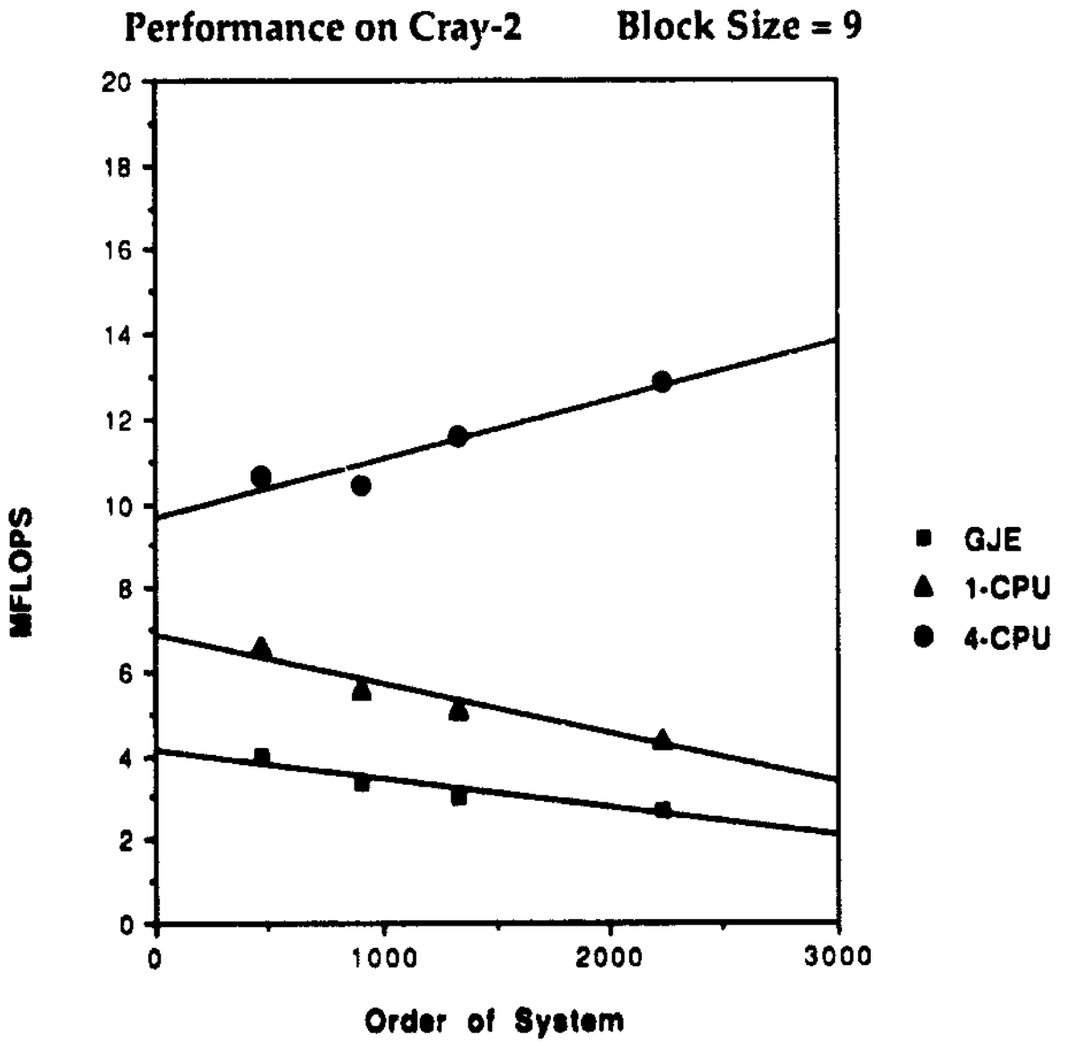

Figure 9
MFLOPS for Blocksize = 9

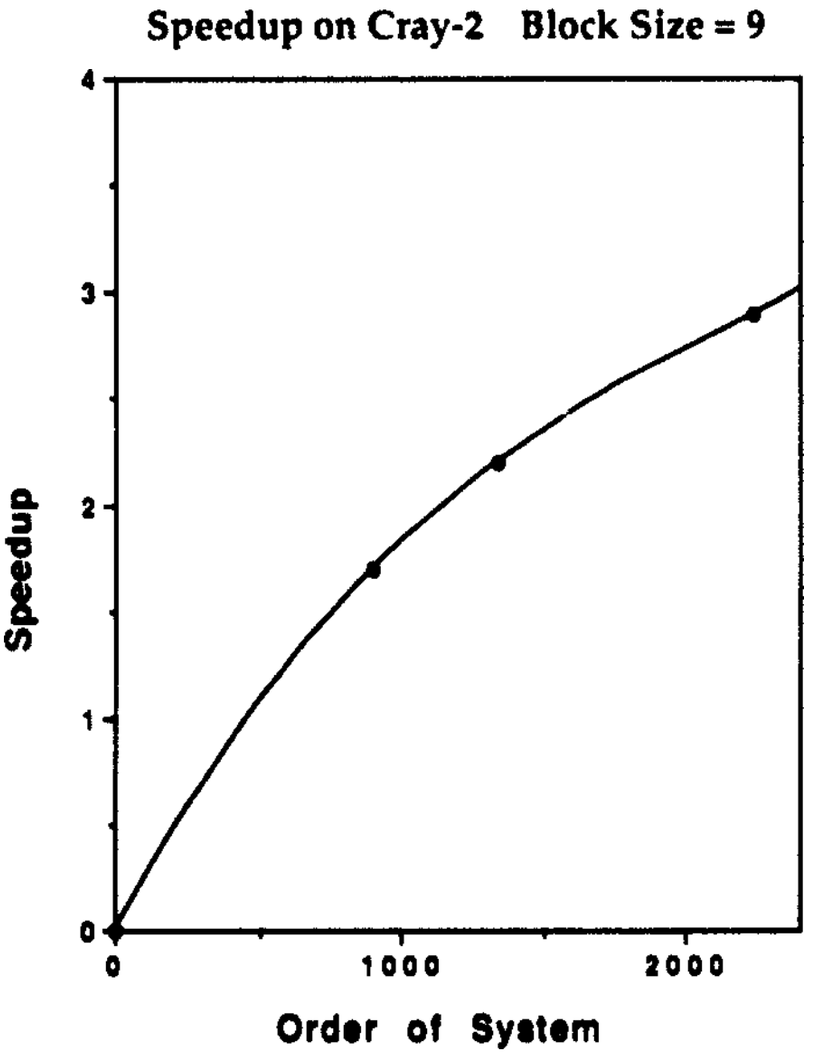

Figure 10
Speedup versus Order for Blocksize = 9

Another result that is somewhat mystifying at first is that the Sameh algorithm on one CPU actually ran faster than did normal Gauss-Jordan elimination. This was not expected since the Sameh algorithm has additional tasks to do, such as solving the reduced matrix, that normal Gauss-Jordan elimination does not. However, a closer examination of the two algorithms reveals a fundamental difference. As the block solver moves down the diagonal, it pivots to produce zeroes everywhere above and below the diagonal in the column. Since the Jacobian is in block form, it need only go down to the bottom edge of the block, since everything below that is already zero. However, it must go all the way to the top of the entire matrix to produce zeroes since fill-in will have occurred during the operations performed on previous columns. As it proceeds down along the diagonal, by the time it reaches the bottom, it is pivoting all the way up the entire length of the matrix. On the other hand, with the Sameh algorithm, the farthest it must pivot would be up to the top of the section, which in this case would be no more than one quarter of the whole matrix. This fundamental difference results in many fewer calculations that the Sameh algorithm must execute to achieve the same results as the normal Gauss-Jordan algorithm. The Sameh algorithm, even on one processor, is a clearly more efficient algorithm. The speedup of the one CPU Sameh algorithm over the normal Gauss-Jordan algorithm is shown in Figure 11 and is essentially constant with the order of the system, always between 1.6 and 1.7.

Figure 12 shows wallclock times for a different block size (21). Again, the Sameh algorithm was successful in significantly speeding up the operation of the program, reaching speedups of greater than three for the largest orders (Figure 13). In fact, if one compares Figures 13 and 10, one will

see that for a given order of the system, the program achieves a greater speedup with the larger block size. However, the program requires a longer overall wallclock time with the larger size, outweighing the increased speedup. The net effect is that an increased block size results in increased speedup, but slower total execution time for a given order of the system.

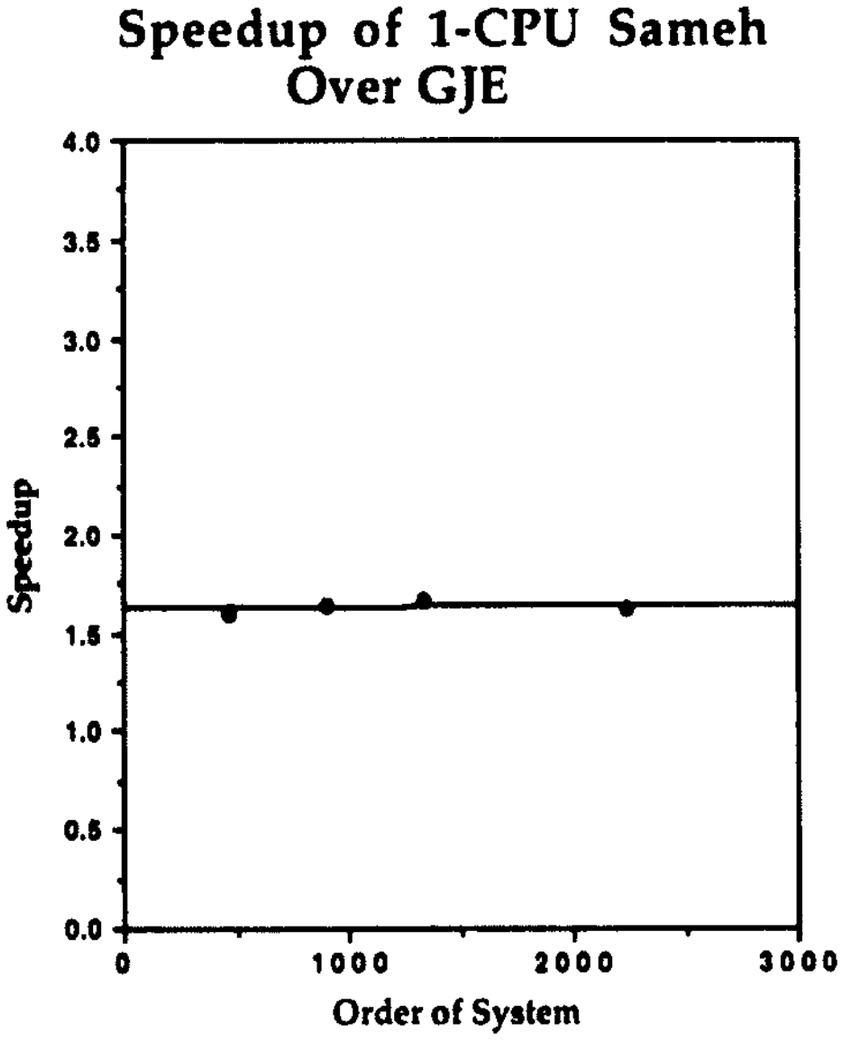

Figure 11
Speedup of Sameh Algorithm over GJE

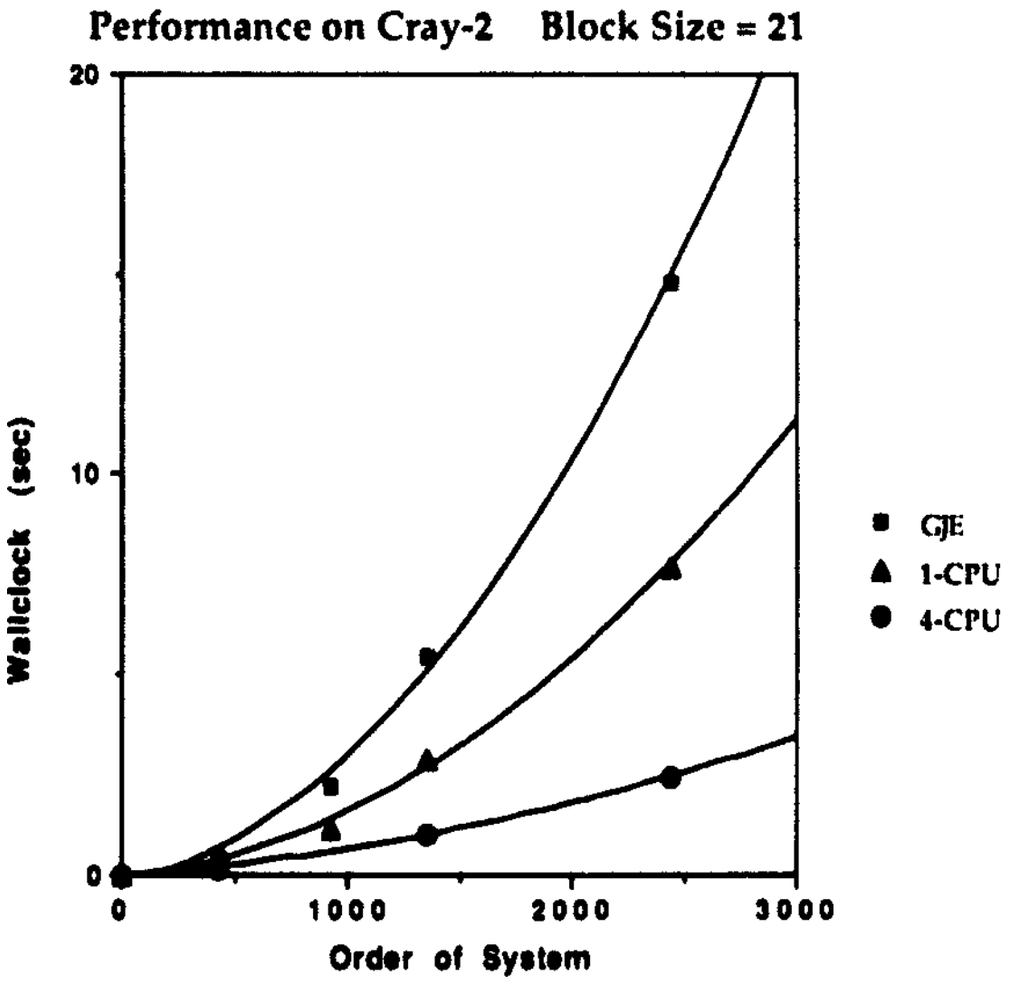

Figure 12
Wallclock Results for Blocksize = 21

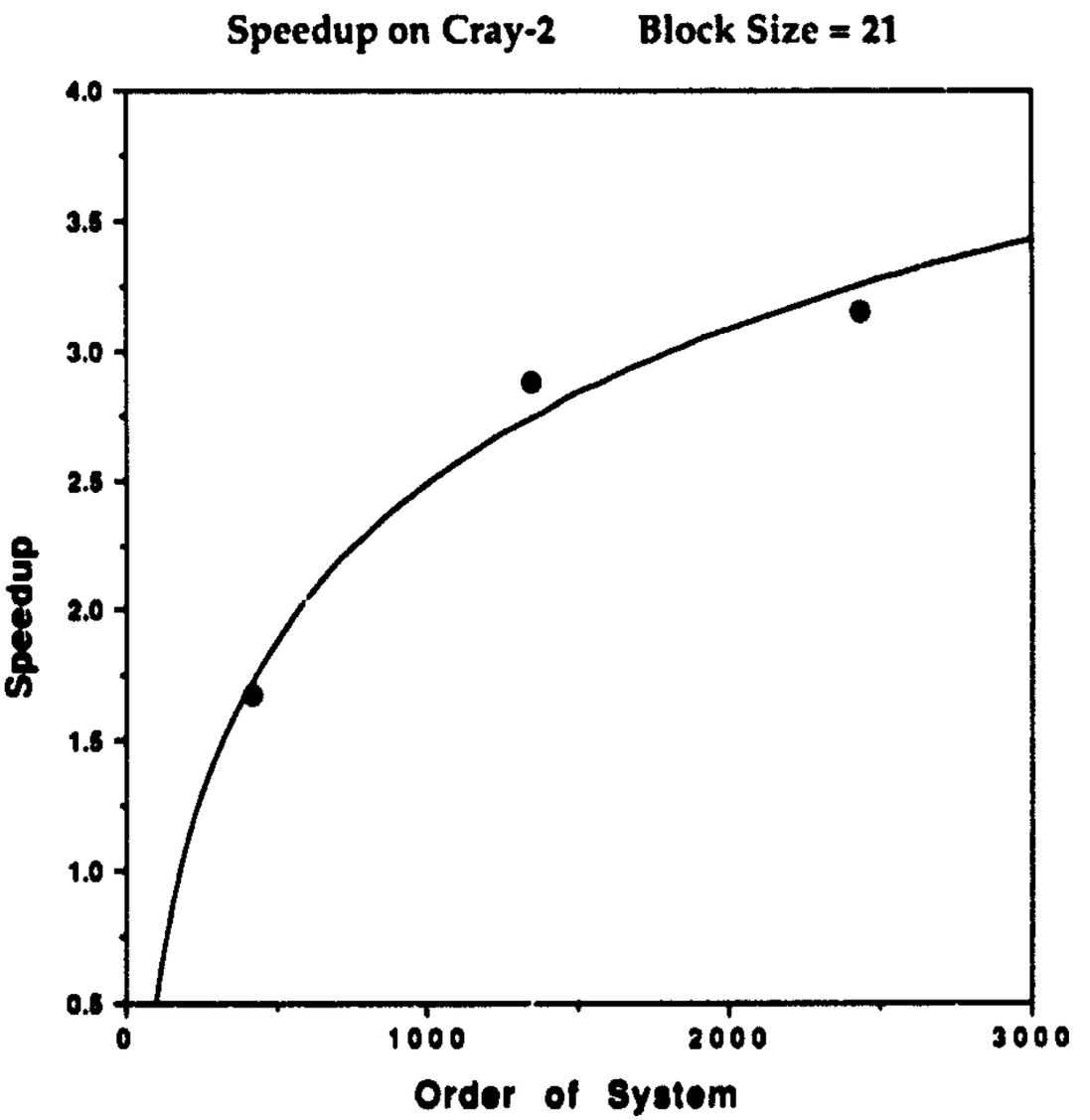

Figure 13
Speedup versus Order for Blocksize = 21

CONCLUSIONS AND RECOMMENDATIONS

The following conclusions can be drawn about the Sameh algorithm as applied to single separation columns:

(1) The Sameh algorithm on four CPU's is capable of achieving speedups in excess of a factor of 3 over normal one CPU operation. Speedup increases with the order of the system and with the block size of the problem.

(2) The Sameh algorithm on one CPU is actually more efficient than normal Gauss-Jordan elimination by a factor of about 1.7.

(3) The Sameh algorithm is robust in that it can handle any number of trays, components, and variables, and can solve both distillation and absorber problems.

As for further study, the following recommendations can be made:

(1) Further runs could be made with even greater system orders (5000 to 10000 variables) in order to determine the asymptotic speedup limit for a given blocksize.

(2) Further attempts can be made to parallelize the remaining subroutines of the program in order to push the maximum speedup closer to 4.

(3) The program can be run on other computer systems (such as the Alliant) which offer larger numbers of processors (64 or more). Runs varying

the number of processors available could determine at what point adding additional processors becomes counterproductive (the size of the reduced matrix becomes too large).

Clearly, the Sameh algorithm is an effective, robust method for solving single separation columns much more quickly than non-parallel algorithms. The next area of research for this topic now will be to attempt to apply this algorithm to systems of interlinked columns. Interlinked systems do not have a true block tri-diagonal structure due to the presence of off diagonal blocks where the columns are connected to each other. The presence of off diagonal blocks will effect the way in which the load must be divided among the available processors, and for some problems the load perhaps may not be efficiently distributed at all. It will be interesting to see if this algorithm can be adopted to handle these more complex systems, because if it can, it offers the potential for greatly increasing the speed of execution of these types of problems.

APPENDIX A

COMPUTER PROGRAM

```
C     program SEPARATE is designed to simulate absorber and
C     distillation columns and is optimized for the Cray-2
C
C     author:  Daniel J. Kaiser


      program SEPARATE

      real x(-4:473),c(468,-9:477),f(468),n(52),y(20),fd(468),
     +  s(468),yv(4),bv(4),yl(4),bl(4),data(2),xnew(468),sums,tol
     +  ,ctemp(30,30),ftemp(30),xtemp(30)

      integer iter,mxiter,i,j,m,k,tray,cp,v,choice,cvar,part,dist(7)

C     v  is the number of variables
C     cp is the number of components
C     tray is the number of trays
C     cvar is the size of the post-Sameh condensed matrix
C        and is equal to (cp+1)*6
C
C     x(-cp,v+cp+1)    is the vector of variable values
C     c(v,-2*cp-1:v+2*cp+1)    is the Jacobian matrix
C     f(v)    is the vector of discrepancy functions
C     n(tray) is a vector containing tray efficiencies
C     y(5*cp) is a vector of K values.  The K-values are
C           to be fitted to a 4th order polynomial:
C            K = AT**4 + BT**3 + CT**2 + DT + E
C           y(1) = A for component 1
C           y(2) = B for component 1
C              etc.
C           y(5) = E for component 1
C           y(6) = A for component 2
C              etc.
C
C     fd(v)    is a vector of feedstream flows
C     s(v)     is a vector of sidestream flows

C     the liquid and vapor enthalpies are fit to straight lines:
C           HL + YV*T + BL
C           HV = YV*T + BV
C     YV(cp), YL(cp), BV(cp), BL(cp) hold these parameters
C     data (2)  Holds user options data
C     sum       is the sum of the squares of f
C     xnew(v)   is the vector of delta x
```

```
C    ctemp(cvar,cvar)  is the reduced Jacobian
C    ftemp(cvar) is the reduced discrepancy vector
C    xtemp(cvar) is the reduced delta x vector
C    dist(7)     contains user option information

     parameter (v = 468, trav = 52, cp = 4, cvar = 30)

     iter = 0

C     open input and output files

      open(unit = 70, file = 'data.dat', status = 'old')
      open(unit = 60, file = 'dist.dat', status = 'old')
      open(unit = 20, file = 'input.dat', status = 'old')
      open(unit = 45, file = 's.dat', status = 'old')
      open(unit = 55, file = 'fd.dat', status = 'old')
      open(unit = 30, file = 'dist1.dat', status = 'old')
      open(unit = 95, file = 'x.out', status = 'new')

C     read in all the input data

      call input(n,y,yv,yl,bv,bl,dist,data,mxiter,s,fd,x,choice)

C     compiler directive to get all four CPUS

CMICS GETCPUS 4

C    Set up initial discrepancy vector

     call discrep(f,x,s,fd,n,y,bv,bl,yv,yl,dist,data)

C    Set up initial derivative matrix

 10  call deriv(f,x,s,fd,n,y,c,bv,bl,yv,yl,dist,data)

C    Decide if user wants to solve using regular block Gaussian
C     elimination (REGSOLVE) or parallel elimination (PARTIAL)

     if (choice .eq. 1)then
      call regsolve(c,f,xnew)
      goto 200
     endif

C    If parallel mode is selected, then do a partial, parallel
C     Gaussian elimination
```

```
      call partial(c,f)

C     select the lines of the system that border the processor
C     divisions

      do 12 part = 1,3
        call select(f,c,ctemp,ftemp,part)
 12   continue

C     solve the reduced matrix

      call fstsolve(ctemp,ftemp,xtemp)

C     perform back substitution

      do 13 part = 1,3
        call scatter1(xtemp,xnew,part)
 13   continue

      call scatter2(f,c,xtemp,xnew)

C     find the best fractional part of the delta x's

 200  call search(x,xnew,fd,s,f,n,y,bv,bl,yv,yl,dist,data)

C     make sure no variables are in a physically meaningless
C     region

      call realvar(x)

C     determine the sum of the squares of the discrepancy
C     functions

      call discrep(f,x,s,fd,n,y,bv,bl,yv,yl,dist,data)

      call sum (f,sums)

C     determine if the method has converged

      iter = iter + 1
      if (iter .eq. mxiter .or. sums .lt. tol) then
       goto 20
      else
       goto 10
      endif
```

```
C    print out final results

 20  call out(x,sums,iter)

     stop
     end


C
C     subroutine DISCREP evaluates the discrepancy
C     functions using the current values for the variables
C


      subroutine DISCREP(f,x,s,fd,n,y,bv,bl,yv,yl,dist,data)

      integer i,j,m,tray,cp,dist(7),opt1,opt2,v
      real f(468),x(-4:473),n(52),y(20),bv(4),bl(4),data(2),
     +  s(468),fd(468),yv(4),yl(4),ev,el,evp,ell,qv,ql,qvp,vptotc,
     +   vtotc,vtotr,ltotc,ltotr,sum,cev,cel,cevp,rev,rel,rell,fsum
      parameter (v = 468, tray = 52, cp = 4, cvar = 30)

C      dimensioning information
C
C     f(v)
C     x(-cp:v+cp+1)
C     n(tray)
C     y(5*cp)
C     bv(cp),bl(4),yv(4),yl(4)
C     data(2)
C     s(v),fd(v)

      opt1 = 0
      opt2 = 0
C     Determine if the problem is and absorber or distillation
C     column.  If absorber, then goto 5

      if (dist(1) .eq. 0) then
       goto 5
      endif

C     This section does the condenser and reboiler
C     opt1 and opt2 are set to one when there are reboilers and
C     condensers.  That way, they main loop knows not to evaluate
C     the first and last trays in the normal fashion.
```

```
      opt1 = 1
      opt2 = 1

C     Evaluate the energy line for the condenser and reboiler
C     based on what user options have been selected

      if (dist(4) .eq. 1) then
         if (dist(2) .eq. 1) then

          cev = 0.0
          cel = 0.0
          cevp = 0.0
          fsum = 0.0
          do 10 i = 1,cp
          cev = cev + (x(i)+s(i))*(yl(i)*x(cp+1)+bl(i))
          cel = cel + (x(i+cp+1)+s(i+cp+1))*(yl(i)*x(cp+1)+bl(i))
            cevp = cevp + x(i+2*cp+1)*(yv(i)*x(3*cp+2)+bv(i))
            fsum = fsum + fd(i)*(yv(i)*fd(cp+1)+bv(i))
     +              + fd(i+cp+1)*(yl(i)*fd(cp+1)+bl(i))
   10     continue

         f(cp+1) = -(cev + cel - cevp - fsum + data(1))

          else

          cev = 0.0
          cel = 0.0
          cevp = 0.0
          fsum = 0.0
          do 20 i = 1,cp
            cev = cev + (x(i)+s(i))*(yv(i)*x(cp+1)+bv(i))
          cel = cel + (x(i+cp+1)+s(i+cp+1))*(yl(i)*x(cp+1)+bl(i))
            cevp = cevp + x(i+2*cp+1)*(yv(i)*x(3*cp+2)+bv(i))
            fsum = fsum + fd(i)*(yv(i)*fd(cp+1)+bv(i))
     +              + fd(i+cp+1)*(yl(i)*fd(cp+1)+bl(i))
   20     continue

          f(cp+1) = -(cev + cel -cevp - fsum + data(1))

         endif
      endif

      if (dist(5) .eq. 1) then
         If (dist(3) .eq. 1) then
```

```
      rev = 0.0
      rel = 0.0
      rell = 0.0
      fsum = 0.0
      do 30 i = 1,cp
        rev = rev + (x(v-2*cp-1+i)+s(v-2*cp-1+i))
     +          *(yv(i)*x(v-cp)+bv(i))
        rel = rel + (x(v-cp+i)+s(v-cp+i)) (yv(i)*x(v-cp)+bv(i))
        rell = rell + x(v-3*cp-1+i)*(yl(i)*x(v-3*cp-1)+bl(i))
        fsum = fsum + fd(v-2*cp-1+i)*(yv(i)*fd(v-cp)+bv(i))
     +          + fd(v-cp+i)*(yl(i)*fd(v-cp)+bl(i))
30    continue

      f(v-cp) = -(rev + rel - rell - fsum - data(2))

    else

      rev = 0.0
      rel = 0.0
      rell = 0.0
      fsum = 0.0
      do 40 i = 1,cp
        rev = rev + (x(v-2*cp-1+i)+s(v-2*cp-1+i))
     +          *(yv(i)*x(v-cp)+bv(i))
        rel = rel + (x(v-cp+i)+s(v-cp+i))*(yl(i)*x(v-cp)+bl(i))
        rell = rell + x(v-3*cp-1+i)*(yl(i)*x(v-3*cp-1)+bl(i))
        fsum = fsum + fd(v-2*cp-1+i)*(yv(i)*fd(v-cp)+bv(i))
     +          + fd(v-cp+i)*(yl(i)*fd(v-cp)+bl(i))
40    continue

      f(v-cp) = -(rev + rel - rell - fsum - data(2))

      endif

    endif

    vtotc = 0.0
    ltotc = 0.0
    vtotr = 0.0
    ltotr = 0.0
    vptotc = 0.0
    do 50 i = 1,cp
      vtotc = vtotc + x(i)
      ltotc = ltotc + x(cp+1+i)
     vtotr = vtotr + x(v-2*cp-1+i)
```

```
      ltotr = ltotr + x(v-cp+i)
      vptotc = vptotc + x(2*cp+1+i)
50    continue

      if(dist(4) .eq. 2) then
       f(cp+1) = -(ltotc - vtotc*data(1))
      endif

      if (dist(4) .eq. 3) then
          f(cp+1) = -(vtotc - data(1))
      endif

      If (dist(4) .eq. 4) then
       f(cp+1) = -(x(dist(6)) - data(1))
      endif

      if (dist(4) .eq. 5) then
       f(cp+1) = -(x(cp+1) - data(1))
      endif

      if(dist(5) .eq. 2) then
        f(v-cp) = -(vtotr - ltotr*data(2))
      endif

      if (dist(5) .eq. 3) then
          f(v-cp) = -(ltotr - data(2))
      endif

      If (dist(5) .eq. 4) then
          f(v-cp) = -(x(v-cp+dist(7)) - data(2))
      endif

      if (dist(5) .eq. 5) then
       f(v-cp) = -(x(v-cp) - data(2))
      endif


C     evaluate the mass and equilibrium lines for the
C     condenser and reboiler based upon user options

      if(dist(2) .eq. 1) then

         sum = 0.0
       do 60 i = 1,cp
           sum = sum + (y(5*(i-1)+1)*x(cp+1)**4
```

```
     +          + y(5*(i-1)+2)*x(cp+1)**3
     +          + y(5*(i-1)+3)*x(cp+1)**2
     +          + y(5*(i-1)+4)*x(cp+1)
     +          + y(5*(i-1)+5))
     +          *x(cp+1+i)/ltotc
60   continue


      do 70 i = 1,cp
       f(i) = -(x(i) + x(i+cp+1) - x(i-cp) - x(i+2*cp+1)
     +      - fd(i) - fd(i+cp+1) + s(i) + s(i+cp+1))
      f(i+cp+1) = -(x(i+cp+1) - ltotc*x(i)/vtotc)
70   continue

     f(cp+2) = -(1 - sum)

    else

      do 80 i = 1,cp
       f(i) = -(x(i) + x(i+cp+1) - x(i-cp) - x(i+2*cp+1)
     +      - fd(i) - fd(i+cp+1) + s(i) + s(i+cp+1))
       f(i+cp+1) = -(n(1)*(y(5*(i-1)+1)*x(cp+1)**4
     +             + y(5*(i-1)+2)*x(cp+1)**3
     +             + y(5*(i-1)+3)*x(cp+1)**2
     +             + y(5*(i-1)+4)*x(cp+1)
     +             + y(5*(i-1)+5))
     +          *vtotc*x(i+cp+1)/ltotc
     +          - x(i) + (1-n(1))*x(i+2*cp+1)*vtotc/vptotc)

80   continue

    endif

    if(dist(3) .eq. 1) then

       sum = 0.0
     do 90 i = 1,cp
        sum = sum + (y(5*(i-1)+1)*x(v-cp)**4
     +          + y(5*(i-1)+2)*x(v-cp)**3
     +          + y(5*(i-1)+3)*x(v-cp)**2
     +          + y(5*(i-1)+4)*x(v-cp)
     +          + y(5*(i-1)+5))
     +          *vtotr/x(v-2*cp-1+i)
90   continue
```

```
      do 100 i = 1,cp
         f(v-2*cp-1+i)=-(x(v-2*cp-1+i)+x(v-cp+i)-x(v-3*cp-1+i)-x(v+i)
     +          -fd(v-2*cp-1+i)-fd(v-cp+i)+s(v-2*cp-1+i)+s(v-cp+i))
        f(v-cp+i)=-(x(v-2*cp-1+i) - vtotr*x(v-cp+i)/ltotr)
100     continue

      f(v-cp+1) = -(1 - sum)

      else

        do 110 i = 1,cp
          f(v-2*cp-1+i)=-(x(v-2*cp-1+i)+x(v-cp+i)-x(v-3*cp-1+i)-x(v+i)
     +          -fd(v-2*cp-1+i)-fd(v-cp+i)+s(v-2*cp-1+i)+s(v-cp+i))
          f(v-cp+i)=-(n(tray)*(y(5*(i-1)+1)*x(v-cp)**4
     +                + y(5*(i-1)+2)*x(v-cp)**3
     +                + y(5*(i-1)+3)*x(v-cp)**2
     +                + y(5*(i-1)+4)*x(v-cp)
     +                + y(5*(i-1)+5))
     +              *vtotr*x(v-cp+i)/ltotr
     +              - x(v-2*cp-1+i))
110     continue

      endif


      if (dist(4) .eq. 2 .or. dist(4) .eq. 3 .or. dist(4) .eq. 4)then
        dump = f(cp+1)
        f(cp+1) = f(cp+2)
        f(cp+2) = dump
      endif

      if (dist(5) .eq. 2 .or. dist(5) .eq. 3 .or. dist(5) .eq. 4)then
        dump = f(v-cp)
        f(v-cp) = f(v-cp+1)
        f(v-cp+1) = dump
      endif



C     this is the main loop that evaluates the trays.  If the problem
C     is an absorber, then opt1 and opt2 will be zero and it will
C     iterate from 1 to tray.  Otherwise, it will iterate from
C     2 to tray-1.

5     do 120 j = 1+opt1,tray-opt2
```

```
      ev = 0.0
        el = 0.0
        evp = 0.0
        ell = 0.0
        fsum = 0.0
        qv = 0.0
        qvp = 0.0
        ql = 0.0
        m = (j-1)*(2*cp+1)
        do 130 i = 1,cp

          ev = ev + (x(i+m)+s(i+m))*(yv(i)*x(m+cp+1) + bv(i))
          el = el + (x(i+m+cp+1)+s(i+m+cp+1))*(yl(i)*x(m+cp+1)+bl(i))
          evp = evp + x(i+m+2*cp+1)*(yv(i)*x(m+3*cp+2) + bv(i))
          ell = ell + x(i+m-cp)*(yl(i)*x(m-cp) + bl(i))
          fsum = fsum + fd(i+m)*(yv(i)*fd(m+cp+1)+bv(i))
     +              + fd(i+m+cp+1)*(yl(i)*fd(m+cp+1)+bl(i))

          qv = qv + x(i+m)
          ql = ql + x(i+m+cp+1)
          qvp = qvp + x(i+m+2*cp+1)

  130   continue

        f(m+cp+1) = -(el + ev - fsum - ell - evp)

        do 140 i = 1,cp

          f(i+m) = -(x(i+m)+x(i+m+cp+1)-x(i+m-cp)-x(i+m+2*cp+1)
     +          -fd(i+m)-fd(i+m+cp+1)+s(i+m)+s(i+m+cp+1))
          f(i+m+cp+1)=-(n(j)*(y(5*(i-1)+1)*x(m+cp+1)**4
     +                + y(5*(i-1)+2)*x(m+cp+1)**3
     +                + y(5*(i-1)+3)*x(m+cp+1)**2
     +                + y(5*(i-1)+4)*x(m+cp+1)
     +                + y(5*(i-1)+5))
     +             *qv*x(i+m+cp+1)/ql
     +             - x(i+m) + (1-n(j))*x(i+m+2*cp+1)*qv/qvp)

  140   continue

  120 continue

        return
        end
```

```
C     subroutine DERIV evaluates the Jacobian at the present
C     values for the variables.

C     the compiler directive CMIC$ MICRO designates this
C     subroutine as one in which microtasking will occur

CMIC$ MICRO
      subroutine DERIV(f,x,s,fd,n,y,c,bv,bl,yv,yl,dist,data)

      integer i,j,k,l,m,o,tray,cp,v,dist(7),opt1,opt2
      real f(468), x(-4:473), c(468,-9:477), n(52), y(20),
     +     bv(4), bl(4), yv(4), yl(4), sum, bbl, bbv, qbl,
     +     qbv, qbvp, option, data(2), s(468), fd(468),dump(18)
      parameter (v = 468, tray = 52, cp = 4, cvar = 30)

C     dimensioning information
C
C     f(v),s(v),fd(v)
C     x(-cp:v+cp+1)
C     c(v,-2*cp-1:v+2*cp+1)
C     n(tray)
C     y(5*cp)
C     bv(4),bl(4),yv(4),yl(4)
C     data(2)
C     dump(4*cp+2)

C     again determine if the problem is an absorber or distillation
C     problem

      opt1 = 0
      opt2 = 0

      if (dist(1) .eq. 0) then
         goto 5
      endif

C     Condenser

      opt1 = 1
      opt2 = 1

C     B-matrix

      if (dist(4) .eq. 1) then
```

```
      if (dist(2) .eq. 1) then

        do 10 j = 1,cp
          c(cp+1,j) = yl(j)*x(cp+1) + bl(j)
          c(cp+1,j+cp+1) = yl(j)*x(cp+1) + bl(j)
10      continue

        bbl = 0.0
        bbv = 0.0

        do 20 i = 1,cp
          bbl = bbl + (x(i+cp+1)+s(i+cp+1))*yl(i)
          bbv = bbv + (x(i)+s(i))*yl(i)
20      continue

        c(cp+1,cp+1) = bbl+bbv

      else

        do 30 j = 1,cp
          c(cp+1,j) = yv(j)*x(cp+1) + bv(j)
          c(cp+1,cp+1+j) = yl(j)*x(cp+1) + bl(j)
30      continue

        bbl = 0.0
        bbv = 0.0
        do 40 i = 1,cp
          bbl = bbl + (x(i+cp+1)+s(i+cp+1))*yl(i)
          bbv = bbv + (x(i)+s(i+cp+1))*yv(i)
40      continue
        c(cp+1,cp+1) = bbl+bbv

      endif

      elseif (dist(4) .eq. 2) then

       do 50 j = 1,cp
          c(cp+1,j) = -data(1)
50      continue

       c(cp+1,cp+1) = 0.0

        do 60 j = cp+2,2*cp+1
          c(cp+1,j) = 1.0
60      continue
```

```
      elseif(dist(4) .eq. 3) then

       do 70 j = 1,cp
          c(cp+1,j) = 1.0
70      continue

       do 80 j = cp+1,2*cp+1
          c(cp+1,j) = 0.0
80      continue

      elseif (dist(4) .eq. 4) then

         do 90 j = 1,cp
        if ( j .eq. dist(6)) then
            c(cp+1,j) = 1.0
        else
            c(cp+1,j) = 0.0
        endif
90      continue

         do 100 j = cp+1,2*cp+1
          c(cp+1,j) = 0.0
100     continue

      elseif (dist(4) .eq.5) then

         do 110 j = 1,2*cp+1
          c(cp+1,j) = 0.0
110     continue

         c(cp+1,cp+1) = 1.0

      endif


    do 120 j = 1,cp
     do 130 i = 1,cp
       if(i .eq. j) then
        c(i,j) = 1.0
       else
        c(i,j) = 0.0
       endif
130    continue
120   continue
```

```
   do 140 i = 1,cp
    c(i,cp+1) = 0.0
140  continue

   do 150 j = cp+2,2*cp+1
    do 160 i = 1,cp
      if((j-cp-1) .eq. i) then
       c(i,j) = 1.0
      else
       c(i,j) = 0.0
      endif
160   continue
150  continue

   if (dist(2) .eq. 1) then

       qbl = 0.0
       qbvp = 0.0
       qbv = 0.0

       do 170 i = 1,cp
        qbl = qbl + x(i+cp+1)
        qbvp = qbvp + x(i+2*cp+1)
        qbv = qbv + x(i)
170    continue

       do 180 j = 1,cp
        c(cp+2,j) = 0.0
180    continue

       do 190 j = 1,cp
        do 200 i = cp+3,2*cp+1
          if ((i-cp-1) .eq. j) then
            c(i,j) = -qbl*(qbv-x(i-cp-1))/(qbv*qbv)
       else
            c(i,j) = -qbl*x(i-cp-1)/(qbv*qbv)
       endif
200     continue
190    continue

       sum = 0.0
       do 210 i = 1,4
         sum = sum - (4*y(5*(i-1)+1)*x(cp+1)**3
  +           + 3*y(5*(i-1)+2)*x(cp+1)**2
  +           + 2*y(5*(i-1)+3)*x(cp+1)
```

```
     +          +   y(5*(i-1)+4))
     +         *x(i+cp+1)/qbl
210  continue

     c(cp+2,cp+1) = sum

    do 215 i = 2,cp
      c(cp+1+i,cp+1) = 0.0
215  continue

     sum = 0.0
     do  220 j = cp+2,2*cp+1
      sum = sum + (y(5*(j-cp-2)+1)*x(cp+1)**4
     +          +  y(5*(j-cp-2)+2)*x(cp+1)**3
     +          +  y(5*(j-cp-2)+3)*x(cp+1)**2
     +          +  y(5*(j-cp-2)+4)*x(cp+1)
     +          +  y(5*(j-cp-2)+5))
     +           *x(j)/(qbl*qbl)
220  continue

     do 230 j = cp+2,2*cp+1
     c(cp+2,j) = (y(5*(j-cp-2)+1)*x(cp+1)**4
     +           + y(5*(j-cp-2)+2)*x(cp+1)**3
     +           + y(5*(j-cp-2)+3)*x(cp+1)**2
     +           + y(5*(j-cp-2)+4)*x(cp+1)
     +           + y(5*(j-cp-2)+5))
     +           *(x(j)-qbl)
     +           /(qbl*qbl) + sum -
     +           (y(5*(j-cp-2)+1)*x(cp+1)**4
     +           + y(5*(j-cp-2)+2)*x(cp+1)**3
     +           + y(5*(j-cp-2)+3)*x(cp+1)**2
     +           + y(5*(j-cp-2)+4)*x(cp+1)
     +           + y(5*(j-cp-2)+5))
     +           *x(j)/(qbl*qbl)
230  continue

     do 240 j = cp+2,2*cp+1
      do 250 i = cp+3,2*cp+1
       if (i .eq. j) then
        c(i,j) = 1-x(i-cp-1)/qbv
     else
        c(i,j) = -x(i-cp-1)/qbv
     endif
250   continue
240  continue
```

```
      else
         qbl = 0.0
       qbvp = 0.0
       qbv = 0.0

         do 260 i = 1,cp
          qbl = qbl + x(i+cp+1)
          qbvp = qbvp + x(i+2*cp+1)
          qbv = qbv + x(i)
260      continue

         do 270 j = 1,cp
          do 280 i = cp+2,2*cp+1

            if((i-cp-1) .eq. j)then
             option = 1.0
         else
             option = 0.0
         endif

            c(i,j) = n(1)*(y(5*(i-cp-2)+1)*x(cp+1)**4
     +              + y(5*(i-cp-2)+2)*x(cp+1)**3
     +              + y(5*(i-cp-2)+3)*x(cp+1)**2
     +              + y(5*(i-cp-2)+4)*x(cp+1)
     +              + y(5*(i-cp-2)+5))
     +              *x(i)/qbl - option + (1-n(1))
     +             *x(i+cp)/qbvp

280       continue
270      continue

         do 290 i = 1,cp
         c(i+cp+1,cp+1) = n(1)*(4*y(5*(i-1)+1)*x(cp+1)**3
     +                   +3*y(5*(i-1)+2)*x(cp+1)**2
     +                   +2*y(5*(i-1)+3)*x(cp+1)
     +                   + y(5*(i-1)+4))
     +                   *qbv*x(i+cp+1)/qbl
290      continue

         do 300 j = cp+2,2*cp+1
          do 310 i = cp+2,2*cp+1

            if(i .eq. j)then
              option = qbl*n(1)*(y(5*(i-cp-2)+1)*x(cp+1)**4
     +                   + y(5*(i-cp-2)+2)*x(cp+1)**3
```

```
     +                  + y(5*(i-cp-2)+3)*x(cp+1)**2
     +                  + y(5*(i-cp-2)+4)*x(cp+1)
     +                  + y(5*(i-cp-2)+5))
     +                  *qbv

          else
              option = 0.0
          endif

             c(i,j) = (option-n(1)*(y(5*(i-cp-2)+1)*x(cp+1)**4
     +                  + y(5*(i-cp-2)+2)*x(cp+1)**3
     +                  + y(5*(i-cp-2)+3)*x(cp+1)**2
     +                  + y(5*(i-cp-2)+4)*x(cp+1)
     +                  + y(5*(i-cp-2)+5))
     +                  *qbv*x(i))/(qbl*qbl)

310       continue
300       continue

       endif


C      C matrix


       if (dist(4) .eq. 1) then

          do 320 j = 1,cp
            c(cp+1,2*cp+1+j) = -(yv(j)*x(3*cp+2)+bv(j))
320       continue

          sum = 0.0
        do 330 i = 1,cp
            sum = sum - x(2*cp+1+i)*yv(i)
330       continue

         c(cp+1,3*cp+2) = sum

       else

          do 340 j = 1,cp+1
            c(cp+1,2*cp+1+j) = 0.0
340       continue

       endif
```

```
      do 350 j = 1,cp
       do 360 i = 1,cp
         If(i .eq. j)then
           c(i,2*cp+1+j) = -1.0
      else
           c(i,2*cp+1+j) = 0.0
      endif
360    continue
350   continue

   if (dist(2) .eq. 0) then

      do 370 i = cp+2,2*cp+1
       do 380 j = 1,cp

         if((i-cp-1) .eq. j)then
          option = qbvp*(1-n(1))*qbv
      else
          option = 0.0
      endif

         c(i,2*cp+1+j) = (option - (1-n(1))*x(i+cp)
     +              *qbv)/(qbvp*qbvp)

380    continue
370   continue

   else

      do 390 i = 1,cp
       do 400 j = 1,cp
       c(i,2*cp+1+j) = 0.0
         c(cp+1+i,2*cp+1+j) = 0.0
400    continue
390   continue

   endif

      do 410 i = 1,cp
       c(i,3*cp+2) = 0.0
       c(cp+1+i,3*cp+2) = 0.0
410   continue

      do 420 j = cp+2,2*cp+1
       do 430 i = 1,2*cp+1
```

```
          c(i,2*cp+1+j) = 0.0
430     continue
420   continue


C     reboiler

C     A-matrix

      do 450 j = 1,cp
        do 460 i = 1,2*cp+1
          c(v-2*cp-1+i,j+v-4*cp-2) = 0.0
460     continue
450   continue

    if (dist(5) .eq. 1)then

      sum = 0.0
      do 470 j = 1,cp
        c(v-cp,v-3*cp-1+j) = -(yl(j)*x(v-3*cp-1)+bl(j))
        sum = sum - x(v-j-3*cp-1)*yl(j)
470   continue
      c(v-cp,v-3*cp-1) = sum

    else

      do 471 j = 1,2*cp+1
      c(v-cp,v-4*cp-2+j) = 0.0
471   continue

      endif

      do 480 i = 2,2*cp+1
        c(v-cp-1+i,v-3*cp-1) = 0.0
480   continue

    do 490 j = cp+2,2*cp+1
        do 500 i = cp+2,2*cp+1
          c(v-cp-1+i,j+v-4*cp-2) = 0.0
500     continue
490   continue

      do 510 j = cp+2,2*cp+1
        do 520 i = 1,cp
```

```
          if((j-cp-1) .eq. i)then
            c(v-2*cp-1+i,j+v-4*cp-2) = -1.0
        else
            c(v-2*cp-1+i,j+v-4*cp-2) = 0.0
        endif
520     continue
510   continue


C     B-matrix

    if (dist(5) .eq. 1) then

        if (dist(3) .eq. 1) then

          do 530 j = 1,cp
            c(v-cp,v-2*cp-1+j) = yv(j)*x(v-cp) + bv(j)
            c(v-cp,v-cp+j) = yv(j)*x(v-cp) + bv(j)
530       continue

          bbl = 0.0
          bbv = 0.0

          do 540 i = 1,cp
            bbl = bbl + (x(v-cp+i)+s(v-cp+i))*yv(i)
            bbv = bbv + (x(v-2*cp-1+i)+s(v-2*cp-1+i))*yv(i)
540       continue

          c(v-cp,v-cp) = bbl+bbv

      else

          do 550 j = 1,cp
            c(v-cp,v-2*cp-1+j) = yv(j)*x(v-cp) + bv(j)
            c(v-cp,v-cp+j) = yl(j)*x(v-cp) + bl(j)
550       continue

          bbl = 0.0
          bbv = 0.0

          do 560 i = 1,cp
            bbl = bbl + (x(v-cp+i)+s(v-cp+i))*yl(i)
            bbv = bbv + (x(v-2*cp-1+i)+s(v-cp+i))*yv(i)
560       continue
```

```
        c(v-cp,v-cp) = bbl+bbv

      endif

      elseif (dist(5) .eq. 2) then

       do 570 j = 1,cp
          c(v-cp,v-2*cp-1+j) = 1.0
570     continue

          c(v-cp,cp+1) = 0.0

          do 580 j = cp+2,2*cp+1
            c(v-cp,v-2*cp-1+j) = data(2)
580     continue

      elseif(dist(5) .eq. 3) then

       do 590 j = 1,cp+1
          c(v-cp,v-2*cp-1+j) = 0.0
590     continue

        do 600 j = cp+2,2*cp+1
          c(v-cp,v-2*cp-1+j) = 1.0
600     continue

      elseif (dist(5) .eq. 4) then

          do 610 j = 1,cp
         if ( j .eq. dist(7)) then
            c(v-cp,v-cp+j) = 1.0
         else
            c(v-cp,v-cp+j) = 0.0
         endif
610      continue

          do 620 j = 1,5
            c(v-cp,v-2*cp-1+j) = 0.0
620     continue

      elseif (dist(5) .eq.5) then

          do 630 j = 1,2*cp+1
            c(v-cp,v-2*cp-1+j) = 0.0
630     continue
```

```
        c(v-cp,v-cp) = 1.0

      endif

    do 640 j = 1,cp
     do 650 i = 1,cp
       if(i .eq. j) then
         c(v-2*cp-1+i,v-2*cp-1+j) = 1.0
       else
         c(v-2*cp-1+i,v-2*cp-1+j) = 0.0
       endif
650    continue
640   continue

    do 660 i = 1,cp
      c(v-2*cp-1+i,v-cp) = 0.0
660   continue

    do 670 j = cp+2,2*cp+1
     do 680 i = 1,cp
       if((j-cp-1) .eq. i) then
         c(v-2*cp-1+i,v-2*cp-1+j) = 1.0
       else
         c(v-2*cp-1+i,v-2*cp-1+j) = 0.0
       endif
680    continue
670   continue

    if (dist(3) .eq. 1) then

        qbl = 0.0
        qbv = 0.0

        do 690 i = 1,cp
          qbl = qbl + x(v-cp+i)
          qbv = qbv + x(v 2*cp-1+i)
690     continue

        do 700 j = cp+2,2*cp+1
          c(v-cp+1,v-2*cp-1+j) = 0.0
700     continue

        do 710 j = cp+2,2*cp+1
          do 720 i = cp+3,2*cp+1
            if ((i-1) .eq. j) then
```

```
            c(v-2*cp-1+i,v-2*cp-1+j)=-qbv*(qbl-x(v-2*cp-1+i))
     +                      /(qbl*qbl)
          else
            c(v-2*cp-1+i,v-2*cp-1+j)=-qbv*x(v-2*cp-1+i)/(qbl*qbl)
          endif
720       continue
710       continue

          sum = 0.0
          do 730 i = 1,4
            sum = sum - (4*y(5*(i-1)+1)*x(v-cp)**3
     +           + 3*y(5*(i-1)+2)*x(v-cp)**2
     +           + 2*y(5*(i-1)+3)*x(v-cp)
     +           + y(5*(i-1)+4))
     +             *qbv/x(v-2*cp-1+i)
730       continue

          c(v-cp+1,v-cp) = sum

        do 740 i = 2,cp
          c(v-cp+i,v-cp) = 0.0
740       continue
          sum = 0.0
          do 750 j = 1,cp
            sum = sum + (y(5*(j-1)+1)*x(v-cp)**4
     +           + y(5*(j-1)+2)*x(v-cp)**3
     +           + y(5*(j-1)+3)*x(v-cp)**2
     +           + y(5*(j-1)+4)*x(v-cp)
     +           + y(5*(j-1)+5))
     +             /x(v-2*cp-1+j)
750       continue

          do 760 j = 1,cp
          c(v-cp+1,v-2*cp-1+j) = (y(5*(j-1)+1)*x(v-cp)**4
     +                    + y(5*(j-1)+2)*x(v-cp)**3
     +                    + y(5*(j-1)+3)*x(v-cp)**2
     +                    + y(5*(j-1)+4)*x(v-cp)
     +                    + y(5*(j-1)+5))
     +                     *(qbv-x(v-2*cp-1+j))
     +                     /(qbv*qbv) + sum -
     +                      (y(5*(j-1)+1)*x(v-cp)**4
     +                     + y(5*(j-1)+2)*x(v-cp)**3
     +                     + y(5*(j-1)+3)*x(v-cp)**2
     +                     + y(5*(j-1)+4)*x(v-cp)
```

```
     +                + y(5*(j-1)+5))
     +                 /x(v-2*cp-1+j)
760  continue

     do 770 j = 1,cp
      do 780 i = cp+3,2*cp+1
       if (i-cp-2 .eq. j) then
         c(v-2*cp-1+i,v-2*cp-1+j) = 1-x(v-2*cp-1+i)/qbl
     else
         c(v-2*cp-1+i,v-2*cp-1+j) = -x(v-2*cp-1+i)/qbl
     endif
780   continue
770  continue

   else

      qbl = 0.0
    qbvp = 0.0
    qbv = 0.0

     do 790 i = 1,cp
      qbl = qbl + x(v-cp+i)
      qbvp = qbvp + x(v+i)
      qbv = qbv + x(v-2*cp-1+i)
790  continue

     do 800 j = 1,cp
      do 810 i = cp+2,2*cp+1

       if((i-cp-1) .eq. j)then
        option = 1.0
     else
        option = 0.0
     endif

        c(v-2*cp-1+i,v-2*cp-1+j)=n(1)*(y(5*(i-cp-2)+1)*x(v-cp)**4
     +                  + y(5*(i-cp-2)+2)*x(v-cp)**3
     +                  + y(5*(i-cp-2)+3)*x(v-cp)**2
     +                  + y(5*(i-cp-2)+4)*x(v-cp)
     +                  + y(5*(i-cp-2)+5))
     +                *x(v-2*cp-1+i)/qbl - option

810   continue
800  continue
```

```
      do 820 i = 1,cp
      c(v-cp+i,v-cp) = n(1)*(4*y(5*(i-1)+1)*x(v-cp)**3
     +                + 3*y(5*(i-1)+2)*x(v-cp)**2
     +                + 2*y(5*(i-1)+3)*x(v-cp)
     +                +   y(5*(i-1)+4))
     +               *qbv*x(v-cp+i)/qbl
  820 continue


      do 830 j = cp+2,2*cp+1
       do 840 i = cp+2,2*cp+1

         if(i .eq. j)then
          option = qbl*n(1)*(y(5*(i-cp-2)+1)*x(v-cp)**4
     +                + y(5*(i-cp-2)+2)*x(v-cp)**3
     +                + y(5*(i-cp-2)+3)*x(v-cp)**2
     +                + y(5*(i-cp-2)+4)*x(v-cp)
     +                + y(5*(i-cp-2)+5))
     +                  *qbv
      else
          option = 0.0
      endif

         c(v-2*cp-1+i,v-2*cp-1+j)=(option-n(1)*
     +                 (y(5*(i-cp-2)+1)*x(v-cp)**4
     +                + y(5*(i-cp-2)+2)*x(v-cp)**3
     +                + y(5*(i-cp-2)+3)*x(v-cp)**2
     +                + y(5*(i-cp-2)+4)*x(v-cp)
     +                + y(5*(i-cp-2)+5))
     +                 *qbv*x(v-2*cp-1+i))/(qbl*qbl)

  840  continue
  830 continue

    endif

    if(dist(4) .eq. 2 .or. dist(4) .eq. 3 .or. dist(4) .eq. 4)then
     do 831 i = 1,4*cp+2
      dump(i) = c(cp+1,i)
  831  continue
       do 832 i = 1,4*cp+2
        c(cp+1,i) = c(cp+2,i)
  832  continue
       do 833 i = 1,4*cp+2
        c(cp+2,i) = dump(i)
```

```
833    continue
   endif

   if(dist(5) .eq. 2 .or. dist(5) .eq. 3 .or. dist(5) .eq. 4)then
    do 834 i = 1,4*cp+2
      dump(i) = c(v-cp,v-4*cp-2+i)
834    continue
       do 835 i = 1,4*cp+2
         c(v-cp,v-4*cp-2+i) = c(v-cp+1,v-4*cp-2+i)
835    continue
       do 836 i = 1,4*cp+2
         c(v-cp+1,v-4*cp-2+i) = dump(i)
836    continue
   endif

5   sum = 0.0

C     this is the main loop of the program.  a few explanations are
C     in order here.  First, note CMIC$ DO GLOBAL.  This tells the
C     compiler that every iteration of loop 85 is independent of
C     each other.  therfore, it should divide the iterations of the
C     loop among the four processors.  Also, note that the counter
C     from 1,52 rather than 1+Opt1,1-opt2.  This is done because
C     the compiler can't microtask the loop if the number of iterations
C     is a variable.  It has to be some constant.  (This particular
C     problem had 52 trays)

CMIC$ DO GLOBAL

C     loop 850 from 1 to tray

    do 850 k = 1,52

C     A-matrix

        m = (k-1)*(2*cp+1)

        do 870 j = 1,cp
         do 880 i = 1,2*cp+1
           c(m+i,m+j-2*cp-1) = 0.0
880      continue
870     continue

        sum = 0.0
        do 890 j = 1,cp
```

```
        c(m+cp+1,m+j-cp) = -(yl(j)*x(m-cp)+bl(j))
        sum = sum - x(m+j-cp)*yl(j)
890   continue

      c(m+cp+1,m-cp) = sum

      do 900 i = 1,cp
        c(m+i,m-cp) = 0.0
900   continue


    do 910 j = cp+1,2*cp+1
       do 920 i = cp+2,2*cp+1
         c(m+i,m+j-2*cp-1) = 0.0
920    continue
910   continue

      do 930 j = cp+2,2*cp+1
       do 940 i = 1,cp
         if((j-cp-1) .eq. i)then
           c(m+i,m+j-2*cp-1) = -1.0
       else
           c(m+i,m+j-2*cp-1) = 0.0
       endif
940    continue
930   continue

860   sum = 0.0

850  continue

CC    B-matrix

CMIC$ DO GLOBAL

C     loop 950 from 1 to tray

    do 950 k = 1,52

       m = (k-1)*(2*cp+1)

       do 960 j = 1,cp
         c(m+cp+1,m+j) = yv(j)*x(m+cp+1) + bv(j)
         c(m+cp+1,m+j+cp+1) = yl(j)*x(m+cp+1) + bl(j)
960    continue
```

```
      bbl = 0.0
      bbv = 0.0

      do 970 i = 1,cp
        bbl = bbl + (x(m+i+cp+1)+s(m+i+cp+1))*yl(i)
        bbv = bbv + (x(m+i)+s(m+i))*yv(i)
970   continue

      c(m+cp+1,m+cp+1) = bbl+bbv


    do 980 j = 1,cp
       do 990 i = 1,cp
         if(i .eq. j)then
           c(m+i,m+j) = 1.0
       else
           c(m+i,m+j) = 0.0
         endif
990    continue
980   continue

    do 1000 i = 1,cp
       c(m+i,m+cp+1) = 0.0
1000  continue

    do 1010 j = cp+2,2*cp+1
       do 1020 i = 1,cp
         if((j-cp-1) .eq. i)then
           c(m+i,m+j) = 1.0
       else
           c(m+i,m+j) = 0.0
       endif
1020    continue
1010   continue

      qbl = 0.0
    qbvp = 0.0
    qbv = 0.0

      do 1030 i = 1,cp
       qbl = qbl + x(m+i+cp+1)
       qbvp = qbvp + x(m+i+2*cp+1)
       qbv = qbv + x(m+i)
1030   continue
```

```
      do 1040 j = 1,cp
       do 1050 i = cp+2,2*cp+1

         if((i-cp-1) .eq. j)then
          option = 1.0
      else
          option = 0.0
      endif

         c(m+i,m+j) = n(k)*(y(5*(i-cp-2)+1)*x(m+cp+1)**4
     +              + y(5*(i-cp-2)+2)*x(m+cp+1)**3
     +              + y(5*(i-cp-2)+3)*x(m+cp+1)**2
     +              + y(5*(i-cp-2)+4)*x(m+cp+1)
     +              + y(5*(i-cp-2)+5))
     +             *x(m+i)/qbl - option + (1-n(k))
     +             *x(m+i+cp)/qbvp

1050    continue
1040   continue

      do 1060 i = 1,cp
       c(m+i+cp+1,m+cp+1) = n(k)*(4*y(5*(i-1)+1)*x(m+cp+1)**3
     +                   + 3*y(5*(i-1)+2)*x(m+cp+1)**2
     +                   + 2*y(5*(i-1)+3)*x(m+cp+1)
     +                   + y(5*(i-1)+4))
     +                  *qbv*x(m+i+cp+1)/qbl
1060   continue

      do 1070 j = cp+2,2*cp+1
       do 1080 i = cp+2,2*cp+1

         if(i .eq. j)then
           option = qbl*n(k)*(y(5*(i-cp-2)+1)*x(m+cp+1)**4
     +                 + y(5*(i-cp-2)+2)*x(m+cp+1)**3
     +                 + y(5*(i-cp-2)+3)*x(m+cp+1)**2
     +                 + y(5*(i-cp-2)+4)*x(m+cp+1)
     +                 + y(5*(i-cp-2)+5))
     +                  *qbv
      else
          option = 0.0
      endif

         c(m+i,m+j) = (option-n(k)*(y(5*(i-cp-2)+1)*x(m+cp+1)**4
     +                   + y(5*(i-cp-2)+2)*x(m+cp+1)**3
     +                   + y(5*(i-cp-2)+3)*x(m+cp+1)**2
```

```
     +                    + y(5*(i-cp-2)+4)*x(m+cp+1)
     +                   + y(5*(i-cp-2)+5))
     +                     *qbv*x(m+i))/(qbl*qbl)

1080     continue
1070    continue

950  continue

C    C matrix

C    loop 1090 from 1 t.    ay

CMIC$ DO GLOBAL
    do 1090 k = 1,52

        m = (k-1)*(2*cp+1)

        do 1110 j = 1,cp
          c(m+cp+1,m+2*cp+1+j) = -(yv(j)*x(m+3*cp+2)+bv(j))
1110    continue

        sum = 0.0
      do 1120 i = 1,cp
          sum = sum - x(m+2*cp+1+i)*yv(i)
1120    continue

       c(m+cp+1,m+3*cp+2) = sum

        do 1130 j = 1,cp
         do 1140 i = 1,cp
           If(i .eq. j)then
             c(m+i,m+2*cp+1+j) = -1.0
        else
             c(m+i,m+2*cp+1+j) = 0.0
        endif
1140      continue
1130    continue

        do 1150 i = cp+2,2*cp+1
         do 1160 j = 1,cp

           if((i-cp-1) .eq. j)then
             option = qbvp*(1-n(k))*qbv
        else
```

```
          option = 0.0
        endif
          c(m+i,m+2*cp+1+j) = (option - (1-n(k))*x(m+i+cp)
     +              *qbv)/(qbvp*qbvp)

1160     continue
1150    continue

        do 1170 i = 1,cp
         c(m+i,m+3*cp+2) = 0.0
         c(m+cp+1+i,m+3*cp+2) = 0.0
1170    continue

        do 1180 j = cp+2,2*cp+1
         do 1190 i = 1,2*cp+1
           c(m+i,m+2*cp+1+j) = 0.0
1190     continue
1180    continue

1090  continue

1100  sum = 0.0

C     Fill in remaining zeroes

CMIC$ DO GLOBAL

C     loop from 1 to tray-2

      do 1200 l = 1,50

          m = 2*cp+2 + l*(2*cp+1)
          o = l*(2*cp+1)

          do 1210 j = m,m+2*cp
           do 1220 i = 1,o
            c(i,j) = 0.0
1220       continue
1210      continue

          do 1230 j = 1,o
           do 1240 i = m,v
            c(i,j) = 0.0
1240       continue
1230      continue
```

```
1200  continue

      return
      end



C
C     subroutine SEARCH finds the best fractional part of
C     the delta x vector.
C

      subroutine SEARCH(x,xnew,fd,s,f,n,y,bv,bl,yv,yl,dist,data)

      real x(-4:473),xnew(468),f(468),n(52),y(20),bv(4),
     +     bl(4),data(2),yv(4),yl(4),x1(-4:473),x2(-4:473),
     +     x3(-4:473),x4(-4:473),fd(468),s(468),
     +     f1(468),f2(468),f3(468),f4(468),sum1,sum2,
     +     sum3,sum4,best1,best2,best,current1,current2

      integer i,v,cp,dist(7)

      parameter (v = 468, tray = 52, cp = 4, cvar = 30)


C     dimensioning information
C
C     x(-cp:v+cp+1)
C     xnew(v),fd(v),s(v)
C     f1(v),f2(v),f3(v),f4(v)
C     y(5*cp)
C     bv(4),bl(4),yv(4),yl(4)
C     data(2)
C     x1(-cp:v+cp+1)
C     x2(-cp:v+cp+1)
C     x3(-cp:v+cp+1)
C     x4(-cp:v+cp+1)
C     the subroutine does not do a full-blown line search,
C     but rather tests four different fractions: 1.00, 0.50
C     0.25, and 0.125.


C     multiply the delta x vector by each fraction tested

      do 10 i = 1,v
```

```
      x1(i) = x(i) + 1.00*xnew(i)
      x2(i) = x(i) + .50*xnew(i)
      x3(i) = x(i) + .25*xnew(i)
      x4(i) = x(i) + .125*xnew(i)

10    continue

C     insert any absorber feed streams(which are not variables)

      do 11 i = -cp,0

      x1(i) = x(i)
      x2(i) = x(i)
      x3(i) = x(i)
      x4(i) = x(i)

11    continue

      do 12 i = v+1,v+cp+1
      x1(i) = x(i)
      x2(i) = x(i)
      x3(i) = x(i)
      x4(i) = x(i)
12    continue

C     evaluate the discrepancy functions with each fractional
C     update

      call discrep(f1,x1,s,fd,n,y,bv,bl,yv,yl,dist,data)
      call discrep(f2,x2,s,fd,n,y,bv,bl,yv,yl,dist,data)
      call discrep(f3,x3,s,fd,n,y,bv,bl,yv,yl,dist,data)
      call discrep(f4,x4,s,fd,n,y,bv,bl,yv,yl,dist,data)

C     calculate the sum of the squares of the discrepancy
C     functions for each fraction

      call sum (f1,sum1)
      call sum (f2,sum2)
      call sum (f3,sum3)
      call sum (f4,sum4)

C     determine which fraction produced the smallest sum
C     of the squares
```

```
      if (sum1 .lt. sum2) then
         current1 = sum1
         best1 = 1.0
      else
         current1 = sum2
         best1 = 0.50
      endif

      if (sum3 .lt. sum4) then
         current2 = sum3
         best2 = 0.25
      else
         current2 = sum4
         best2 = 0.125
      endif


      if (current1 .lt. current2)then
         best = best1
      else
         best = best2
      endif

C     update the variable vector with the best fracitonal
C     part of the delta x vector

      do 20 i = 1,v
         x(i) = x(i) + best*xnew(i)
 20   continue

      return
      end


C
C     subroutine SUM calculates the sum of the squares
C     of the discrepancy functions
C

      subroutine SUM(f,sums)

      real f(468),sums

      integer v,i
```

```
      parameter (v = 468, tray = 52, cp = 4, cvar = 30)

C     dimensioning information
C
C     f(v)

      sums = 0.0
      do 10 i = 1,v
         sums = sums + f(i)*f(i)
 10   continue

      return
      end


C     subroutine REALVAR checks all the variables to see
C     if they have passed into physically meaningless
C     ranges.  If they have, then they are changed to
C     at least be physically feasible.
C
C     note:  this subroutine may require alteration
C            from problem to problem depending on
C            what you want the minimum feasible
C           tray temperature to be


      subroutine REALVAR(x)

      real x(-4:473)

      integer v,cp,tray,i,m,j

      parameter (v = 468, tray = 52, cp = 4, cvar = 30)

C     dimensioning information
C
C     x(-cp:v+cp+1)

      do  10 j = 1,tray
        m = (j-1)*(2*cp+1)

          do  20 i = m+1,m+cp

C     check the gas flowrates.  if any have become negative
C     set to zero
```

```
            if (x(i) .lt. 0.0) then
              x(i) = 0.0
            endif

C     check the temperatures.  if any have fallen below a certain
C     minimum temperature, set them to that minimum.  (for this
C     particular problem, the minimum was 0.0)

            if (x(m+cp+1) .lt. 0.0) then
              x(m+cp+1) = 0.0
            endif

C     check the liquid flowrates.  if any have become negative
C     set to zero

            if (x(cp+1+i) .lt. 0.0) then
              x(cp+1+i) = 0.0
            endif

 20     continue
 10   continue
      return
      end


C     subroutine INPUT reads in all the input data from the
C     data files.  for information on how to construct the
C     data files, please see the associated README document

      subroutine INPUT(n,y,yv,yl,bv,bl,dist,data,mxiter,s,fd,x,choice)

      real n(52),y(20),yv(4),yl(4),bv(4),bl(4),data(2),
     +     s(468),fd(468),x(-4:473)
      integer v,tray,cp,i,dist(7),choice,mxiter
      parameter (v = 468, tray = 52, cp = 4, cvar = 30)

C     dimensioning information
C
C     n(tray)
C     y(5*cp)
C     yv(4),yl(4),bv(4),bl(4)
C     data(2)
C     s(v)
```

```
C    fd(v)
C    x(-cp:v+cp+1)

C    read in tray efficiencies

     read(20,*) (n(i), i=1,tray)

C    read in equilibrium curve fit parameters

     read(20,*) (y(i), i=1,5*cp)

C    read in enthalpy curve fit parameters

     do 100 i = 1,cp
       read(20,*) yv(i)
       read(20,*) yl(i)
       read(20,*) bv(i)
       read(20,*) bl(i)
 100   continue

C    read in user options

     read(60,*) (dist(i), i = 1,7)
     read(70,*) data(1)
     read(70,*) data(2)
     read(70,*) mxiter
     read(70,*) choice

C    read in sidestream and feedstream specifications

     read(45,*) (s(i), i=1,v)
     read(55,*) (fd(i), i=1,v)

C    Read in initial guesses for variables

     read(30,*) (x(i), i=-cp,v+cp+1)

     return
     end


C    subroutine OUT writes the final results to
C    the output file x.out
C
```

```
      subroutine OUT(x,sums,iter)

      real x(-4:473),sums
      integer v,tray,cp,iter
      parameter (v = 468, tray = 52, cp = 4, cvar = 30)

C     dimensioning information
C
C     x(-cp:v+cp+1)

      write(95,*) sums
      write(95,*) iter

C     tops

      do 21 i = 1,cp+1
        write(95,*) x(i)
 21   continue

C     bottoms

      do 22 i = v-cp,v
        write(95,*) x(i)
 22   continue

      return
      end


C     subroutine REGSOLVE performs normal, routine block
C     gaussian elination.  The user may elect this option or
C     the parallel, Sameh option


      subroutine REGSOLVE (c,f,x)

      integer v,tray,cvar,s,i,j,k,inp,part,cp,limit
      real f(468),c(468,-9:477),x(468),dp,dump(468),elem1,elem2
      parameter (v = 468, tray = 52, cp = 4, cvar = 468)

C     dimensioning information
C
C     f(v),x(v),dump(v)
C     c(v,-2*cp-1:v+2*cp+1)
C
```

```
C     NOTE ** cvar in the parameter statement should be set to
C     the total number of variables (v) for this subroutine only

C     work on just the first column

      elem1 = c(1,1)

      do 1000 j = 1,4*cp+2
        c(1,j) = c(1,j)/elem1
1000  continue

      f(1) = f(1)/elem1

      do 10 i = 2,2*cp+1
         elem2 = c(i,1)
         do 20 j = 1,4*cp+2
           c(i,j) = c(i,j) - elem2*c(1,j)
20       continue
         f(i) = f(i) - elem2*f(1)
10    continue

C     work on all the other columns except the last

      do 25 k = 2,cvar-1

C     check to see if the diagonal element is very small
C     must replace a diagonal element tha  is very small.
C     First, check the one right below it.  If it is greater
C     than 0.1, go ahead and switch rows.  If it is not, keep
C     going down the column until one is found.  If it gets to
C     the bottom without finding one greater than 0.1, it will
C     just take the best one it could find.

      if (abs(c(k,k)) .lt. .0001)then

C     start scanning down the column to find a better element.

       j = k+1
       inp = k

       if((k+2*cp+1) .gt. cvar) then
         limit = cvar
       else
          limit = k+2*cp+1
       endif
```

```
C     if it found one greater than .1, goto 16 and switch rows
C     otherwise keep looking

 17     if (abs(c(j,k)) .gt. .1)then
        inp = j
        goto 16
       endif

C     keep track of the best one found to this point

       if (abs(c(j,k)) .gt. abs(c(inp,k)))then
        inp = j
       endif

       if (j .lt. limit)then
        j = j+1
        goto 17
       endif

C     switch rows

 16     do 11 i = 1,cvar
        dump(i) = c(k,i)
 11     continue

       do 12 i = 1,cvar
        c(k,i) = c(inp,i)
 12     continue

       do 13 i = 1,cvar
        c(inp,i) = dump(i)
 13     continue

       dp = f(k)
       f(k) = f(inp)
       f(inp) = dp

      endif

C      perform the operations necessary to get zeroes above
C      and below the diagonal element

C      set limits so the program doesn't operate on parts
C      of the matrix that will always be zero
```

```
      elem1 = c(k,k)

       if((k+4*cp+2) .gt. cvar) then
        limit1 = cvar
       else
           limit1 = k+4*cp+2
       endif

       if((k+2*cp+1) .gt. cvar) then
        limit2 = cvar
       else
           limit2 = k+2*cp+1
       endif

       do 30 j = k,limit1
         c(k,j) = c(k,j)/elem1
30     continue

       f(k) = f(k)/elem1

C      fill in all the zeroes above the diagonal

       do 40 i = 1,k-1

        elem 2 = c(i,k)

        do 50 j = k,limit1
          c(i,j) = c(i,j) - elem2*c(k,j)
50      continue

        f(i) = f(i) - elem2*f(k)

40     continue

C      fill in all the zeroes below the column

       do 60 i = k+1,limit2

        elem2 = c(i,k)

        do 70  j = k,limit1
          c(i,j) = c(i,j) - elem2*c(k,j)
70      continue

        f(i) = f(i) - elem2*f(k)
```

```
 60    continue
 25   continue

C     work on the last column

      elem1 = c(cvar,cvar)

      c(cvar,cvar) = c(cvar,cvar)/elem1

      f(cvar) = f(cvar)/elem1

      do 90 i = 1,cvar-1
         elem2 = c(i,cvar)
         c(i,cvar) = c(i,cvar) - elem2*c(cvar,cvar)
         f(i) = f(i) - elem2*f(cvar)
 90   continue

C     store the values in the variable vector

      do 200 i = 1,cvar
       x(i) = f(i)
 200  continue


      return
      end


C     subroutine FSTSOLVE does normal full matrix
C     gaussian elimination on the condensed matrix
C

      subroutine FSTSOLVE (c,f,x)

      integer v,tray,cvar,s,i,j,k,inp,part,cp,limit
      real f(30),c(30,30),x(30),dp,dump(30),elem1,elem2
      parameter (v = 468, tray = 52, cp = 4, cvar = 30)

C     dimensioning information
C
C     dimension all arrays in this subroutine to cvar

C     operate on just the first column

      elem1 = c(1,1)
```

```
      do 1000 j = 1,cvar
        c(1,j) = c(1,j)/elem1
1000  continue

      f(1) = f(1)/elem1

      do 10 i = 2,cvar
         elem2 = c(i,1)
         do 20 j = 1,cvar
           c(i,j) = c(i,j) - elem2*c(1,j)
20       continue
         f(i) = f(i) - elem2*f(1)
10    continue

C     operate on all the other columns except the last

      do 25 k = 2,cvar-1

C     must replace a diagonal element that is very small.
C     First, check the one right below it.  If it is greater
C     than 0.1, go ahead and switch rows.  If it is not, keep
C     going down the column until one is found.  If it gets to
C     the bottom without finding one greater than 0.1, it will
C     just take the best one it could find.

      if (abs(c(k,k)) .lt. .0001)then
       j = k+1
       inp = k

       if((k+2*cp+1) .gt. cvar) then
        limit = cvar
      else
          limit = k+2*cp+1
      endif

17     if (abs(c(j,k)) .gt. .1)then
        inp = j
       goto 16
      endif

      if (abs(c(j,k)) .gt. abs(c(inp,k)))then
        inp = j
      endif
```

```
      if (j .lt. limit)then
       j = j+1
       goto 17
      endif

16     do 11 i = 1,cvar
       dump(i) = c(k,i)
11     continue

      do 12 i = 1,cvar
       c(k,i) = c(inp,i)
12     continue

      do 13 i = 1,cvar
       c(inp,i) = dump(i)
13     continue

      dp = f(k)
      f(k) = f(inp)
      f(inp) = dp

     endif

C     back to the normal column operations

     elem1 = c(k,k)
       do 30 j = k,cvar
        c(k,j) = c(k,j)/elem1
30    continue

      f(k) = f(k)/elem1

C     fill in all the zeroes above the diagonal element

      do 40 i = 1,k-1

       elem 2 = c(i,k)

       do 50 j = k,cvar
        c(i,j) = c(i,j) - elem2*c(k,j)
50     continue

       f(i) = f(i) - elem2*f(k)

40    continue
```

```
C       fill in all the zeroes below the diagonal element

        do 60 i = k+1,cvar

         elem2 = c(i,k)

         do 70  j = k,cvar
           c(i,j) = c(i,j) - elem2*c(k,j)
70       continue

         f(i) = f(i) - elem2*f(k)

60     continue
25    continue

C     work on the last column

      elem1 = c(cvar,cvar)

      c(cvar,cvar) = c(cvar,cvar)/elem1

      f(cvar) = f(cvar)/elem1


      do 90 i = 1,cvar-1
         elem2 = c(i,cvar)
         c(i,cvar) = c(i,cvar) - elem2*c(cvar,cvar)
         f(i) = f(i) - elem2*f(cvar)
90    continue

C     put the values into the variable vector

      do 200 i = 1,cvar
       x(i) = f(i)
200   continue


      return
      end

C     subroutine PARTIAL is the main subroutine of the
C     program.  It does a partial gaussian elimination
C     on the system, reducing it to a diagonal of ones
C     plus some columns of fill-in.  And, it does this
C     in parallel.
```

```
C     CMIC$ MICRO designates this subroutine as a microtasked
C     subroutine

CMIC$ MICRO
      subroutine PARTIAL (c,f)

      real f(468),c(468,-9:477),dp,dump(468),elem1,elem2

      integer v,best,inp,tray,s,i,j,k,part,cp,limit

      parameter (v = 468, tray = 52, cp = 4, cvar = 30)

C      dimensioning information
C
C     f(v),dump(v)
C     c(v,-2*cp-1:v+2*cp+1)

C     This subroutine consists of one giant DO loop which
C     iterates four times (four processors).  Each iteration is
C     independent of the others.  The first performs gaussian
C     elimination on the top quarter of the system, the second
C     iteration on the second quarter, and so on.
C
C     CMIC$ DO GLOBAL designates this loop as a microtasked loop
C     so that all processors may work on it simultaneously


CMIC$ DO GLOBAL
      do 5 part = 1,4

      s = (part-1)*(v/4)

C      work on just the first column

      elem1 = c(s+1,s+1)
      do 1000 j = s-2*cp,s+4*cp+2
        c(s+1,j) = c(s+1,j)/elem1
 1000 continue

      f(s+1) = f(s+1)/elem1

      do 10 i = s+2,s+2*cp+1
         elem2 = c(i,s+1)
         do 20 j = s-2*cp,s+4*cp+2
```

```
        c(i,j) = c(i,j) - elem2*c(s+1,j)
20     continue
       f(i) = f(i) - elem2*f(s+1)
10   continue

C    operate on all the other columns except the last one

     do 25 k = s+2,s+v/4-1

C    must replace a diagonal element that is very small.
C    First, check the one right below it.  If it is greater
C    than 0.1, go ahead and switch rows.  If it is not, keep
C    going down the column until one is found.  If it gets to
C    the bottom without finding one greater than 0.1, it will
C    just take the best one it could find.

       if (abs(c(k,k)) .lt. .0001)then
        j = k+1
        inp = k

        if((k+2*cp+1) .gt. s+v/4) then
         limit = s+v/4
        else
            limit = k+2*cp+1
        endif

17      if (abs(c(j,k)) .gt. .1)then
         inp = j
         goto 16
        endif

        if (abs(c(j,k)) .gt. abs(c(inp,k)))then
         inp = j
        endif

        if (j .lt. limit)then
         j = j+1
         goto 17
        endif

16      do 11 i = 1,v
        dump(i) = c(k,i)
11      continue
```

```
      do 12 i = 1,v
       c(k,i) = c(inp,i)
12     continue

      do 13 i = 1,v
       c(inp,i) = dump(i)
13     continue

      dp = f(k)
      f(k) = f(inp)
      f(inp) = dp

     endif

C     back to normal gaussian operations

      elem1 = c(k,k)

C     set limits so that it doesn't operate on parts
C     of the matrix that will always be zero

      if((k+4*cp+2) .gt. s+v/4) then
      limit1 = s+v/4
     else
        limit1 = k+4*cp+2
     endif

      if((k+2*cp+1) .gt. s+v/4) then
      limit2 = s+v/4
     else
        limit2 = k+2*cp+1
     endif

      do 30 j = k,limit1
       c(k,j) = c(k,j)/elem1
30    continue

      do 31 j = s-2*cp,s
       c(k,j) = c(k,j)/elem1
31    continue

      do 32 j = s+v/4+1,s+v/4+2*cp+1
       c(k,j) = c(k,j)/elem1
32    continue
```

```
      f(k) = f(k)/elem1

C     fill in all the zeroes above the diagonal element

      do 40 i = s+1,k-1

       elem 2 = c(i,k)

       do 50 j = k,limit1
         c(i,j) = c(i,j) - elem2*c(k,j)
50     continue

       do 51 j = s-2*cp,s
         c(i,j) = c(i,j) - elem2*c(k,j)
51     continue

       do 52 j = s+v/4+1,s+v/4+2*cp+1
         c(i,j) = c(i,j) - elem2*c(k,j)
52     continue

       f(i) = f(i) - elem2*f(k)

40    continue

C     fill in all the zeroes below the diagonal element

      do 60 i = k+1,limit2

       elem2 = c(i,k)

       do 70  j = k,limit1
         c(i,j) = c(i,j) - elem2*c(k,j)
70     continue

       do 71  j = s-2*cp,s
         c(i,j) = c(i,j) - elem2*c(k,j)
71     continue

       do 72  j = s+v/4+1,s+v/4+2*cp+1
         c(i,j) = c(i,j) - elem2*c(k,j)
72     continue

       f(i) = f(i) - elem2*f(k)

60    continue
```

```
25   continue

C    operate on the last column

     elem1 = c(s+v/4,s+v/4)

     do 80 j = s+v/4,s+v/4+2*cp+1
        c(s+v/4,j) = c(s+v/4,j)/elem1
80   continue

     do 81 j = s-2*cp,s
        c(s+v/4,j) = c(s+v/4,j)/elem1
81   continue

     f(s+v/4) = f(s+v/4)/elem1

     do 90 i = s+1,s+v/4-1
        elem2 = c(i,s+v/4)

        do 110 j = s+v/4,s+v/4+2*cp+1
          c(i,j) = c(i,j) - elem2*c(s+v/4,j)
110     continue

        do 111 j = s-2*cp,s
          c(i,j) = c(i,j) - elem2*c(s+v/4,j)
111     continue

        f(i) = f(i) - elem2*f(s+v/4)

90   continue

5    continue

     return
     end


C    subroutine SELECT picks out the rows of the system
C    that border the processor divsions and condenses them
C    into a condensed matrix with dimension cvar,cvar

     subroutine SELECT(f,c,ctemp,ftemp,part)

     real f(468), c(468,-9:477), ctemp(30,30), ftemp(30)
```

```
      integer i,j,s,ss,v,cp,tray,cvar,part

      parameter (v = 468, tray = 52, cp = 4, cvar = 30)

C     dimensioning information
C     f(v)
C     c(v,-2*cp-1,v+2*cp+1)
C     ctemp(cvar,cvar)
C     ftemp(cvar,cvar)

      s = (part-1)*(2*cp+2)
      ss = part*v/4 - (cp+1) - (part-1)*(2*cp+2)

      do 100 j = 1,2*cp+2
         do 200 i = s+1,s+2*cp+2
           ctemp(i,j) = c(ss+i,v/4-cp-1+j)
200   continue
100   continue

      do 300 j = 2*cp+3,4*cp+4
       do 400 i = s+1,s+2*cp+2
          ctemp(i,j) = c(ss+i,2*(v/4)-3*cp-3+j)
400   continue
300   continue

      do 500 j = 4*cp+5,6*cp+6
         do 600 i = s+1,s+2*cp+2
         ctemp(i,j) = c(ss+i,3*(v/4)-5*cp-5+j)
600   continue
500   continue

     do 700 i = 1,cvar/3
         ftemp(i+s) = f(i+part*(v/4)-cp-1)
700   continue


      return
      end


C     subroutine SCATTER1 merely takes the variables
C     solved for in FSTSOLVE and scatters them back
C     into the vector of delta x's (xnew)

      subroutine SCATTER1(xtemp,xnew,part)
```

```
      real xtemp(30),xnew(468)

      integer i,part,s,v,cvar,cp

      parameter (v = 468, tray = 52, cp = 4, cvar = 30)

C     dimensioning information
C
C     xtemp(cvar)
C     xnew(v)

      s = (part-1)*(2*cp+2)

      do 100 i = 1,cvar/3
         xnew(i+part*(v/4)-cp-1) = xtemp(i+s)
100   continue

      return
      end

C     subroutine SCATTER2 takes the values obtained by SCATTER1
C     and performs back substitution to calculate the rest of
C     the delta x vector (xnew)
      subroutine SCATTER2(f,c,xtemp,xnew)

      real f(468),c(468,-9:477),xtemp(30),xnew(468)
     +     ,ctot1,ctot2
      integer i,j,v,cp

      parameter (v = 468, tray = 52, cp = 4, cvar = 30)

C     work on first quarter of vector

      ctot1 = 0.0
      do 100 i = 1,v/4-cp-1
         do 200 j = 1,cp+1
         ctot1 = ctot1 - c(i,v/4+j)*xtemp(cp+1+j)
200   continue
      xnew(i) = f(i) + ctot1
      ctot1 = 0.0
100   continue

C     work on second quarter of vector
```

```
      ctot1 = 0.0
      ctot2 = 0.0
      do 300 i = v/4+cp+2,2*v/4-cp-1
         do 400 j = 1,cp+1
           ctot1 = ctot1 - c(i,v/4-cp-1+j)*xtemp(j)
         ctot2 = ctot2 - c(i,2*v/4+j)*xtemp(3*cp+3+j)
400   continue
         xnew(i) = f(i) + ctot1 + ctot2
         ctot1 = 0.0
         ctot2 = 0.0
300   continue

C     work on third quarter of vector

      ctot1 = 0.0
      ctot2 = 0.0
      do 500 i = 2*v/4+cp+2,3*v/4-cp-1
         do 600 j = 1,cp+1
           ctot1 = ctot1 - c(i,2*v/4-cp-1+j)*xtemp(2*cp+2+j)
           ctot2 = ctot2 - c(i,3*v/4+j)*xtemp(5*cp+5+j)
600   continue
         xnew(i) = f(i) + ctot1 + ctot2
         ctot1 = 0.0
         ctot2 = 0.0
500   continue

C     work on fourth quarter of vector

      ctot1 = 0.0
      do 700 i = 3*v/4+cp+2,v
         do 800 j = 1,cp+1
           ctot1 = ctot1 - c(i,3*v/4-cp-1+j)*xtemp(4*cp+4+j)
800   continue
         xnew(i) = f(i) + ctot1
         ctot1 = 0.0
         ctot2 = 0.0
700   continue

      return
      end
```

APPENDIX B

PROBLEM SPECIFICATIONS

ROBUSTNESS TEST PROBLEMS

Absorber 1 atm

trays	20
components	4
variables	130

K values

Component	100°F	200°F
A	500.0	550.0
B	1.50	1.80
C	0.90	1.00
D	1.0×10^{-6}	1.5×10^{-6}

Liquid Enthalpies (10^3 BTU/mole)

A	0.01	0.013
B	0.30	0.33
C	0.40	0.44
D	1.50	1.90

Vapor Enthalpies (10^3 BTU/mole)

A	1.00	1.002
B	1.80	1.82
C	2.00	2.03
D	5.75	5.95

Feeds (moles/time)

Component	Tray 1 (liq) 125°F	Tray 20 (vap) 200°F
A	0.0	75.0
B	0.0	15.0
C	0.0	10.0
D	100.0	0.0

Program output

Sum of squares . . 5.93×10^{12}

	tops	bottoms
A	74.83	0.166
B	4.69	10.31
C	0.0209	9.98
D	9.17×10^{-5}	100.0

Distillation Column 1 atm

trays . . . 20
components . 3
variables . . 140
reflux ratio . . 1000
recovers 80% of feed in bottoms
sidestream on condenser equal to 30% of reflux liquid

K values

Component	100°F	30°F
n-butane	3.27	0.843
i-pentane	1.40	0.245
n-pentane	1.02	0.18

Liquid Enthalpies (BTU/mole)

n-butane	2222	-117
i-pentane	2501	-91
n-pentane	2626	-81

Vapor Enthalpies (BTU/mole)

n-butane	11225	9668
i-pentane	13200	11258
n-pentane	13600	11649

Feed and sidestreams (moles/time)

Component	feed tray 10 liq 30°F	side condenser liq
A	500	298.9
B	500	0.09245
C	500	5.935×10^{-4}

Program output

Sum of squares . . 3.97×10^{-16}

	tops	bottoms
A	0.9997	200.1
B	9.04×10^{-5}	499.9
C	3.61×10^{-6}	500.0

TIMING PROBLEMS

Block Size 9

used absorber shown above -- varied number of trays
times are per iteration

run #1

52 trays
468 variables

results

algorithm	wallclock time (sec)
BGE	0.3375
1-CPU Sameh	0.2135
4-CPU Sameh	0.1325

run #2

100 trays
900 variables

results

algorithm	wallclock time (sec)
BGE	1.258
1-CPU Sameh	0.765
4-CPU Sameh	0.412

run #3

148 trays
1332 variables

results

algorithm	wallclock time (sec)
BGE	2.866
1-CPU Sameh	1.715
4-CPU Sameh	0.756

run #4

248 trays
2232 variables

results

algorithm	wallclock time (sec)
BGE	8.575
1-CPU Sameh	5.279
4-CPU Sameh	1.809

Block Size = 21

modified absorber -- added additional components for a total of 10

K values

Component	100°F	200°F
A	500	550
B	300	350
C	100	150
D	1.5	1.8
E	1.1	1.2
F	0.9	1.0
G	0.5	0.8
H	$1x10^{-5}$	$1.05x10^{-5}$
I	$5x10^{-6}$	$5.5x10^{-6}$
J	$1x10^{-6}$	$1.5x10^{-6}$

Liquid Enthalpies (10^3 BTU/mole)

A	10.0	13.0
B	20.0	23.0
C	30.0	33.0
D	300.0	330.0
E	380.0	420.0
F	400.0	440.0
G	350.0	380.0
H	1000.0	1400.0
I	1200.0	1600.0
J	1500.0	1900.0

Vapor Enthalpies (10^3 BTU/mole)

A	1000.0	1002.0
B	1100.0	1102.0
C	1200.0	1202.0
D	1800.0	1820.0
E	1900.0	1920.0
F	2000.0	2030.0
G	1850.0	1870.0
H	3000.0	3200.0
I	4000.0	4200.0
J	5750.0	5950.0

Feeds (moles/time)

Component	Tray 1 (liq) 125°F	Tray 20 (vap) 200°F
A	0	75
B	0	75
C	0	75
D	0	15
E	0	15
F	0	10
G	0	10
H	100	0
I	100	0
J	100	0

run #1

20 trays
420 variables

results

algorithm	wallclock time (sec)
BGE	0.44867
1-CPU Sameh	0.2775
4-CPU Sameh	0.15427

run #2

44 trays
924 variables

results

algorithm	wallclock time (sec)
BGE	2.218
1-CPU Sameh	1.182
4-CPU Sameh	-------

run #3

64 trays
1344 variables

results

algorithm	wallclock time (sec)
BGE	5.443
1-CPU Sameh	2.848
4-CPU Sameh	0.988

run #4

116 trays
2436 variables

results

algorithm	wallclock time (sec)
BGE	14.874
1-CPU Sameh	7.668
4-CPU Sameh	2.4335

BIBLIOGRAPHY

(1) Naphtali, Leonard M., and Sandholm, Donald P., "Multicomponent Separation Calculations by Linearization", *AIChE Journal*, Vol. 17, 148-153 (1971).

(2) Levesque, John M., and Williamson, Joel W., *A Guidebook to Fortran on Supercomputers*, Academic Press, San Diego, California (1989).

(3) Cray Research Inc., *Multitasking Programmer's Manual*, Cray Research Inc., Mendota Heights, Minnesota (1989).

(4) Sameh, Ahmed, and Berry, Michael W., "Multiprocessor Schemes for Solving Block Tridiagonal Linear Systems", *International Journal of Supercomputer Applications*, Vol. 2, 37-57, (1988).